User's Manual

LOGICAL
VISION
LTD

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_06875

WiT

Version 4.8.5

User's Manual

Logical Vision Ltd.

Copyright ©1995 by Logical Vision Ltd. All Rights Reserved.

WiT, the Logical Vision logo and WiT logo are trademarks of Logical Vision Ltd. WitFlow is a trademark of Logical Vision Ltd. and Datacube Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Printed in Canada.

Contents

1 Introduction **1**

1.1 Igraphs . . . 1

1.2 Extensive Operator Libraries . . . 2

1.3 Generalized Data Objects . . . 3

1.4 Client/Server Model . . . 3

1.5 Parallel Execution . . . 3

1.6 Data/Demand-Driven Modes . . . 4

1.7 CAD-like Environment . . . 4

1.8 User Extendable . . . 5

1.8.1 Adding New Operators . . . 5

1.8.2 Adding New Object Types . . . 5

1.8.3 Adding New Server Types . . . 5

2 Getting Started **7**

2.1 Installation . . . 7

2.1.1 Extract the distribution . . . 7

2.1.2 WITHOME, GHOME, HOSTTYPE and Paths . . . 8

2.2 Basics . . . 9

2.3 GUI Environment . . . 10

2.3.1 The Motif Environment . . . 10

2.3.2 The OpenLook Environment . . . 11

2.3.3 On-line Help . . . 12

2.4 Floating License 12

3 Quick Tour **15**

3.1 Exercise 1: Displaying an Image 15

3.1.1 Building the Display Image Igraph 15

3.1.2 Running the Display Image Igraph 19

3.2 Exercise 2: Studying Images 19

3.3 Exercise 3: Managing Igraphs 19

3.4 Exercise 4: Performing a Linear Series of Operations 20

3.5 Exercise 5: WiT Data Objects 21

3.6 Exercise 6: Working with Frame Grabbers 23

3.7 Exercise 7: Executing Operators in Demand Mode 24

3.8 Exercise 8: Manipulating the Demand Mode Stack 26

3.9 Exercise 9: Advanced Igraph Editing 26

3.9.1 Selecting Igraph Objects 27

3.9.2 Moving Objects 27

3.9.3 Cut, Copy, Paste, and Duplicate 28

3.9.4 Editing Links 28

3.9.5 Icon Appearance 28

3.9.6 Panning and Zooming 28

3.10 Exercise 10: Hierarchical Operators and Icon Editor 29

3.10.1 Creating the Sub-Graph 29

3.10.2 Making the Sub-Graph an Operator 30

3.10.3 Using the Hierarchical Operator 30

3.10.4 Modifying Sub-Graph Behavior 31

3.11 Learning More 31

4 Operators **33**

4.1 Invoking Operators 33

4.1.1 Graphical Operator Library Panels 33

4.1.2 The Operator Selector . . . 35
4.2 Operator Icons . . . 36
4.2.1 Operator Definitions . . . 36
4.2.2 Icon Graphics . . . 37
4.2.3 Modifying Icon Graphics . . . 37
4.2.4 Icon Cache . . . 37
4.2.5 Default Icon . . . 38
4.3 Operator Properties . . . 38
4.4 Getting Help for Operators . . . 38
4.4.1 Getting Help for an Operator Port . . . 38

5 Data Objects **39**
5.1 Viewing Objects . . . 39
5.1.1 Viewing Objects in Data-Driven Mode . . . 39
5.1.2 Viewing Objects in Demand-Driven Mode . . . 39
5.1.3 Viewing Object Files . . . 40
5.2 Studying Images . . . 40
5.2.1 Studying Pixel Values . . . 41
5.2.2 Plotting Pixel Profiles . . . 41
5.2.3 Adjusting Image Colors . . . 41
5.3 Resizing Images . . . 41
5.4 Zooming and Panning Images . . . 42
5.5 Image Types . . . 42
5.6 Contrast and Color Control . . . 43
5.6.1 Grayscale Images . . . 43
5.6.2 Color Images . . . 44
5.7 Printing Images . . . 44
5.8 Studying General Objects . . . 44
5.8.1 Scroll Bars . . . 44
5.8.2 Changing Font . . . 44

5.8.3 Saving a Displayed Data Object 44
5.9 Printing a General Data Object 45
5.10 Reading/Writing Objects and Search Paths 45
5.11 Object Window Placement 45

6 Data-Driven Mode **47**
6.1 Running WiT in Data-Driven Mode 47
6.2 Igraphs 48
6.2.1 Creating a New Igraph 48
6.2.2 Reading an Igraph 48
6.2.3 Saving an Igraph 48
6.2.4 Running an Igraph 49
6.2.5 Pausing Igraph Execution 50
6.2.6 Resuming a Paused Igraph 50
6.2.7 Stopping an Igraph 50
6.2.8 Zooming and Panning an Igraph 51
6.2.9 Customizing the Grid 51
6.2.10 Getting Help for Igraphs 51
6.3 Igraph Operators 51
6.3.1 Moving Operators on an Igraph 52
6.3.2 Cutting, Copying, and Pasting Operators 52
6.3.3 Changing Operator Properties 52
6.3.4 Operator Color Indicators 52
6.3.5 Parameters as Inputs 53
6.3.6 Getting Help for Operators in an Igraph 53
6.4 Igraph Links 53
6.4.1 Creating Links 53
6.4.2 Editing Links 54
6.4.3 Changing Link Properties 54
6.4.4 Link Direction 55

6.4.5 Frayed Link Ends . . . 55
6.4.6 Probes . . . 55
6.4.7 Getting Help for Links . . . 55
6.5 Link Junctions . . . 55
6.5.1 Creating a Junction . . . 56
6.5.2 Deleting a Junction . . . 56
6.6 Manipulating Igraph Objects . . . 57
6.6.1 Selecting Igraph Objects . . . 57
6.6.2 Cutting Igraph Objects . . . 57
6.6.3 Copying Objects . . . 57
6.6.4 Pasting Objects . . . 58
6.6.5 Duplicating Objects . . . 58
6.6.6 Pick Tolerance . . . 58
6.6.7 Cut and Paste Across Igraphs . . . 58
6.6.8 Object Names . . . 59
6.7 Hierarchical Operators and Sub-Graphs . . . 59
6.7.1 Making a New Hierarchical Operator . . . 60
6.7.2 Sub-Graph Ports . . . 61
6.7.3 Promoting Parameters . . . 61
6.7.4 Changing Operator Parameters in a Sub-Graph . . . 61
6.7.5 Modifying a Sub-Graph . . . 62
6.7.6 Copy, Paste, and Undo Across Sub-Graphs . . . 62

7 Demand-Driven Mode **63**

7.1 Running WiT in Demand-Driven Mode . . . 63
7.2 Demand-Driven Mode Cards . . . 63
7.2.1 Displaying the Contents of a Card . . . 64
7.2.2 Card Labels . . . 65
7.2.3 Moving Cards . . . 65
7.2.4 Deleting Cards . . . 65

7.2.5 Copying Cards . . . 66
7.2.6 Changing the Number of Available Cards . . . 66
7.3 History Logging . . . 66
7.3.1 Reading a Demand History File . . . 67
7.3.2 Saving a Demand History . . . 67
7.3.3 Running a Demand History . . . 67
7.3.4 Editing a History File . . . 68

8 General Techniques **69**
8.1 Zooming and Panning with the Mouse . . . 69
8.1.1 Zooming . . . 69
8.1.2 Panning . . . 70
8.1.3 Restoring to Original Scale and Position . . . 70
8.2 Resizing Windows . . . 70
8.3 Graphical Objects . . . 71
8.3.1 Selecting Graphical Objects . . . 71
8.3.2 Moving and Adjusting Graphical Objects . . . 71
8.3.3 Adjusting Graphical Objects . . . 72
8.4 Selecting Run Speed . . . 72
8.5 Printing . . . 73
8.5.1 Printer Fonts . . . 73

9 Interactive Operators **75**
9.1 GetData . . . 75
9.1.1 Entering Graphical Objects . . . 75
9.1.2 Selecting, Moving, and Adjusting Objects . . . 76
9.1.3 Edit Menu . . . 76
9.1.4 Fill Patterns . . . 77
9.1.5 Fonts . . . 77
9.1.6 Object Colors . . . 77

9.2 Surface . . . 77
9.2.1 Properties Panel . . . 78
9.3 Graph . . . 79
9.3.1 Input Types . . . 79
9.3.2 Operator Parameters . . . 80
9.3.3 Properties Panel . . . 81
9.3.4 Curve Types . . . 82
9.3.5 Displaying Exact Coordinates . . . 82
9.3.6 Zooming and Panning . . . 82
9.3.7 Multiple Curves on the Same Graph . . . 82
9.4 System . . . 82

10 Parts and Tools List **85**

10.1 The Main Window . . . 85
10.1.1 Main Window Menu Bar . . . 86
10.1.2 Group Properties . . . 87
10.1.3 Tracing . . . 88
10.1.4 Tool Bar . . . 89
10.2 Status Window . . . 90
10.3 Workspace Menus . . . 90
10.3.1 Data-Driven Mode Workspace Menu . . . 90
10.3.2 Demand-Driven Mode Workspace Menu . . . 91
10.4 Options Panels . . . 91
10.4.1 Setup Panel . . . 91
10.4.2 Data-Driven Options . . . 93
10.4.3 Demand-Driven Options . . . 94
10.5 Operator Selector . . . 94
10.6 Operator Keyword Search Panel . . . 95
10.7 File Panel . . . 96
10.8 Choice Lists . . . 97

10.9 Font Panel . . . 97
10.10 Color Panel . . . 98
10.11 Zoom/Pan Panel . . . 99
10.12 Demand-Driven Operator Properties . . . 100
10.12.1 Data-Driven Operator Properties . . . 100
10.13 Link Properties . . . 102
10.14 Array Editor . . . 102
10.14.1 Menu bar . . . 103
10.14.2 Panel . . . 103
10.14.3 Array . . . 103
10.15 Icon Editor . . . 104
10.16 Operator Library Pop-up Menu . . . 105
10.17 Image Pop-up Menu . . . 105
10.18 Image Properties . . . 105
10.18.1 Contrast Ramp Control . . . 108
10.18.2 Image Colormaps . . . 108
10.19 Three-Dimension Viewing Controls . . . 109
10.20 Print Panel . . . 111

11 Server Manager **113**

11.1 Menu Bar . . . 113
11.2 Tool Bar . . . 114
11.3 Connectivity Chart . . . 114
11.4 Status Window . . . 115
11.5 Choosing a Configuration File . . . 115

12 Advanced Igraph Concepts **117**

12.1 Brief Review of Data-Flow Concept . . . 117
12.2 Data-Flow and WiT Igraphs . . . 119
12.3 The Scheduler . . . 119

12.3.1 Data transport . . . 119
12.3.2 Firing Strategy . . . 120
12.3.3 Flat and Hierarchical Scheduling . . . 120
12.4 WiT Extensions to Basic Data-Flow Model . . . 120
12.4.1 Unconnected Output Ports . . . 121
12.4.2 Fire-On-Any Operators . . . 121
12.4.3 Parameter as Input on Fire-On-Any Operators . . . 121
12.4.4 Sync Tokens . . . 121
12.5 Some Caveats of Data-Flow . . . 122
12.5.1 Deadlock . . . 122
12.5.2 Connecting an Operator to Itself . . . 123

13 Configuration **125**

13.1 GUI Initialization . . . 125
13.1.1 Initialization Options List . . . 125
13.1.2 Default Initialization . . . 126
13.1.3 Initialization File . . . 127
13.1.4 Command Line Options . . . 128
13.2 Color . . . 128
13.2.1 Workspace Colors . . . 128
13.3 Server Configuration . . . 130
13.3.1 Choosing a WiT Configuration . . . 130
13.3.2 Creating a New Configuration File . . . 130
13.4 WiT Configuration File Format . . . 130
13.4.1 Server Type Definition . . . 131
13.4.2 Operator List Definition . . . 131
13.4.3 Library Names . . . 132
13.4.4 Object Search Path . . . 132
13.4.5 Igraph Search Path . . . 133
13.4.6 Library Search Path . . . 133

13.4.7 Default Configuration File 133
13.5 Password Information 134
13.6 Servers 134
13.6.1 Multiple server types 135
13.6.2 Multiple Users 136
13.7 Directory Organization 137
13.8 Read/Write Directories 137

A File Formats **111**
A.1 Demand History File Format 111

List of Figures

1.1 Example WiT Session 2

3.1 WiT on Start Up 16
3.2 Igraph for Reading and Displaying an Image 16
3.3 Read/Write Library Panel 17
3.4 RdImage Properties Panel 17
3.5 Interactive Library 18
3.6 Operators 18
3.7 A Series of Operators 20
3.8 Igraph for Demonstrating Use of Data Objects 21
3.9 Data Objects 23
3.10 Igraph for Grabbing an Image 24
3.11 WiT in Demand-Driven Mode 25
3.12 RdImage Operator Properties in Demand-Driven Mode 25
3.13 After Executing Operator getElem 26
3.14 Moving a Card in Demand-Driven Mode 27
3.15 Hierarchical Operator Usage 29

4.1 Point Library Panel 34

5.1 Pixel Profile Plot 42
5.2 An Example of Color Image Processing 43

6.1 Typical Operator Appearance 52

6.2 Example Use of Link Junctions 56
6.3 Make Operator Panel 60

7.1 WiT in Demand-Driven Mode 64

9.1 GetData Panel 76
9.2 Surface Operator Output and Properties 78
9.3 3-D Wireframe 79
9.4 Multi-Data-Set Graph 80
9.5 Graph Properties Panel 81
9.6 Example Use of System Operator 83

10.1 WiT Main Window 85
10.2 Node Group Properties Panel 88
10.3 Link Group Properties Panel 88
10.4 Setup Panel for General Options 92
10.5 Data-Driven Options Panel 93
10.6 Demand-Driven Options Panel 94
10.7 Operator Selector Panel 94
10.8 Operator Keyword Search Panel 95
10.9 File Panel 96
10.10 Font Panel 97
10.11 WiT Color Panel 98
10.12 Zoom/pan Panel 99
10.13 Example Demand-Driven Operator Properties 100
10.14 Example Data-Driven Operator Properties 101
10.15 Link Properties Panel 102
10.16 Array Editor 103
10.17 Example Icon Editor Session 104
10.18 Image Properties Panel 106
10.19 Custom Image Color Panel 109

10.20 Three-Dimension Controls 110
10.21 Three-Dimension Control Terminology 110
10.22 Print Panel 111

11.1 WiT Manager 114
11.2 Server Property Panel 115

12.1 Data Dependency Graph 118
12.2 Example of Deadlock in an Igraph 122
12.3 Deadlock Fixed 123
12.4 Illegal Igraph Constructs 123

13.1 Color Editor for Workspace Colors 129
13.2 Multiple Server Types 135

Chapter 1

Introduction

WiT is a powerful visual programming package for designing computer algorithms with block diagrams which can be *executed*. It is particularly suitable for complex image processing applications in areas such as machine vision, medical imaging, and imaging research. Solutions are rapidly developed and elegantly described by building imaging graphs, called *Igraphs*, using icons and links with "point-and-click" simplicity. Figure 1.1 shows an example of an Igraph and a typical WiT session.

1.1 Igraphs

The Igraph concept is one of the most noticeable features in WiT. An Igraph is a graphical representation of an imaging algorithm. On an Igraph, each icon represents an operation, such as reading an image file, performing a convolution, or generating a lookup table. Each operation takes some number of input objects and produces some output objects, which flow along paths specified by directed links.

Operators are represented by *graphical* icons, not just a block diagram with text inside. This makes your imaging algorithm pleasant to work with and easy to understand. The properties of each operator can be interactively modified and displayed below the icon so that the Igraph is a *complete* representation of your algorithm. Flow control operators such as *if-then-else* and *for-loop* allow you to incorporate conditional logic within an Igraph.

When an Igraph is executed, WiT dynamically shows you data objects traveling along links or being processed by operators, and which inputs and outputs are being consumed and generated. Controls for pausing, continuing, or stopping Igraph execution are available.

Igraphs present a quick way of prototyping and testing image analysis procedures without committing hours of tedious programming. With meaningful graphical icons for operators and parameter values displayed directly below each operator, an Igraph is not only the means of executing an algorithm, it also serves as concise and accurate documentation for it.

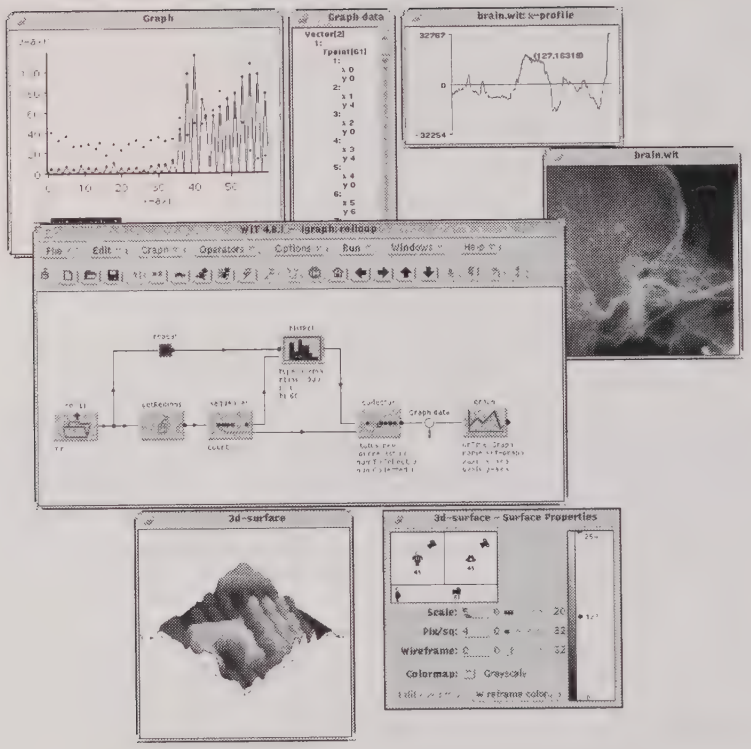

Figure 1.1: Example WiT Session

1.2 Extensive Operator Libraries

WiT comes with a rich set of operators classified into iconified libraries in areas such as read/write, point, filter, transform, morphology, segmentation, and measurement. A powerful flow control library provides operators which allow Igraphs to employ conditional branches, loops, synchronization, and sequencing through arrays of objects.

Operators are available for interactive manipulation and viewing of image data. For example, you can interactively enter graphical data, such as points, rectangles, or polygons on images, plot data sets as graphs in a variety of styles, display image data as 3-D surfaces, adjust image contrast, and display image intensity profiles.

1.3 Generalized Data Objects

The initial stages of imaging algorithms may deal exclusively with images, but as an imaging algorithm progresses to a higher level segmentation, measurement, or recognition phase, other data types such as chain-codes and feature vectors become necessary. All data types, not just images, are treated as *objects* in WiT. An object is similar to the `structure` or `record` used in some programming languages. It can contain many fields. For example, an object for an employee may contain his name, address, position, and salary as fields. A `department` object may contain an array of `employee` objects. An `image` object would contain its width, height, depth, and data bits.

Any object type can be viewed by placing a magnifying glass (Figure 1.1) on a link. All object types can be displayed, read, saved, and printed. For example, an image and a vector of statistical data are being displayed in Figure 1.1.

1.4 Client/Server Model

WiT is based on a client/server model and consists of two parts: a graphical user interface (GUI) and some (zero to infinitely many) computational servers, which are separate programs communicating with one another using some kind of interprocess process communication (IPC) mechanism. The client/server arrangement is superior to a single program in many respects:

- The GUI and server components can run on different computers, which may even be of different makes. For example, you can have the GUI running on a PC, and the server running on a Cray supercomputer.
- You can have more than one server, which together with the data-flow model, will be able to run operators in parallel on different servers at the same time.
- New operators can be added by rewriting the server only, which is small and easy to understand, and compiles fast.
- Because the server is small, it can even run on a diskless computer, which may need to be physically close to special purpose hardware.
- The server is often continually changed to accommodate new operators, which means it is prone to bugs. Since the server is a separate program from the GUI, any bugs are confined to the server itself, which makes it easy to detect errors and debug new operators.

1.5 Parallel Execution

Parallel computation is understandably highly complicated and is still a major research topic. WiT does not solve all the issues in parallel computation, but it offers a practical application environment which allows the user to tap into the power of networked computers which are so common nowadays.

A parallel algorithm which involves running small operations, such as addition and multiplication, is termed *fine grain* parallelism. If the individual operations are large, perhaps each involving the

execution of a program such as a compiler or word processor, then it is termed *coarse grain*. If it is something in between, such as when the operators are function calls, then it is termed *medium grain*. Obviously the boundaries between the different classifications are rather vague. WiT is best suited to medium to coarse grain parallelism. It will work with fine grain operations, but the overhead involved may destroy any potential speed gains.

Parallelism is expressed in an Igraph by having link branches, or operators that produce more than one output. You can have as many branches as you like from a link. This allows you to easily represent parallelism in your algorithm. WiT employs a sophisticated scheduler that analyzes link connections and determine how to dispatch operations efficiently across a network of servers (if available).

An operation is dispatched by sending the necessary data objects to a server and then instructing it to carry out the operation. While this server is busy, another operation can be dispatched to another server, and so on until all available servers are busy. As soon as a server comes back with results, it can be given a new task. The WiT scheduler knows what servers are capable of executing which operators (if the servers are different).

Objects can be sent between heterogeneous machine architectures so that WiT can take advantage of different hardware platforms such as Sparc host processors, image accelerators, or DSP networks. All the necessary data conversion and communication is transparent to the user.

Often the movement of data objects is only a concept. A handle to the data is passed from one operator to another. Actual data transfer between the GUI and the servers is only done when it is absolutely necessary.

1.6 Data/Demand-Driven Modes

Although Igraphs are extremely powerful in expressing complex imaging algorithms, sometimes it is more convenient to try different operators and see the results immediately. For this reason, WiT provides two modes of execution: *data-driven* and *demand-driven*.

Igraphs are used only in data-driven mode. In demand-driven mode, you can run operators on images or objects that you specify, and the results are displayed immediately. This is helpful in exploring and understanding the set of imaging operations WiT offers, as well as debugging any new operators that you have developed yourself. Actions performed in demand mode can be saved as *demand scripts* and played back later.

1.7 CAD-like Environment

WiT is designed to handle highly complex Igraphs. Hundreds of operators can be utilized in a single algorithm without any problems. High-speed zoom and pan features make it possible to view hundreds of operators on the computer screen.

WiT facilitates the design and layout of Igraphs by providing various features which are commonly found in CAD systems. Igraph components may be selected then copied or moved about the

workspace. The workspace may be increased or reduced in size. Links are easily wired or reshaped where vertices are snapped to a grid of variable resolution. Pick accuracy can be set small so that closely spaced objects can be distinguished, or set large so that objects can be easily selected. Groups of operators can be interactively made into sub-graphs, allowing algorithm details to be hidden if the user is not interested. Named ports are used to specify connectivity among Igraphs. Sub-graph nesting level is unlimited. Multiple sub-graphs can be viewed at the same time.

1.8 User Extendable

Users can easily extend WiT by defining new C functions and data types, and custom servers to access specialized hardware. Although WiT was developed for image processing, it can serve as a general visual programming tool for developing solutions to problems interactively using a data-flow model. The object-oriented structure of WiT promotes the design of well structured and well behaved algorithms. Here we take a brief look at the procedures. For details, see the *Programmer's Guide.*

1.8.1 Adding New Operators

Adding your own operators to WiT is easy. All you have to do is:

1. Specify the number and type of inputs, outputs, and parameters that the new operator requires, using a special but very simple *operator definition language.*
2. Write a C function to process the inputs and generate the appropriate outputs,
3. Rebuild the server.

The new operator is then ready for use!

1.8.2 Adding New Object Types

Just about any data structure you can create in the C programming language can be made into a WiT object. Built-in object types include integers, floats, strings, images, and vectors of these basic types.

By using the WiT object definition language similar to the C `struct` syntax, user-defined objects are rapidly defined and built into WiT using the WiT server manager, which is both part of WiT or can be run as a separate program. WiT will then know how to send/receive, free, copy, print, read/write, and display the new object type. For details, see the *Programmer's Guide.*

1.8.3 Adding New Server Types

Due to the client/server nature of WiT, the server code is extremely simple. It is shipped with WiT in source code form, so that you can modify it to create your own server. For example, you may

want to make a server which initializes your specialized hardware on start-up, or you may want to make a server that has a graphical user interface, possibly running on a different display other than the WiT GUI.

Chapter 2

Getting Started

In this chapter, we describe the procedures for installing WiT on your computer. Then we outline some basic techniques you will need in order to explore WiT. Once you have mastered these basic techniques, it is possible to start experimenting by trial-and-error instead of reading the rest of this manual.

2.1 Installation

To install WiT, first choose a directory for where you wish to install the distribution. If many people are going to use WiT, then use a directory which is NFS mountable by everyone. If you are the only user, then a convenient place may be your home directory. If you are upgrading a previous version of WiT, then save your `$WITHOME/config/witPasswd` file to a safe place, extract the software, then move your password file back to the newly installed `config` directory. In this example, we assume that WiT will be installed in shared directory, eg. `/usr/local/wit`.

1. `mkdir /usr/local/wit`
2. `cd /usr/local/wit`

2.1.1 Extract the distribution

The distribution is provided on several floppy disks and can be extracted simply by inserting disk one and following the instructions on the disk labels. In the case of Solaris, it is necessary to become super-user (root) and stop the volume manager so that the floppy drive can be accessed directly from the WiT installation script. To do this, follow the instructions below:

1. Become super-user: `su root`
2. Get volume manager process id. This is the number reported in the first column from the output of this command: `ps -e | grep vold`

3. Stop the vold process: `kill vold-pid`
4. Now insert floppy disk one and follow label instructions.
5. When you're all done, you may restart the volume manager by typing:

```
/usr/sbin/vold &.
```

2.1.2 WITHOME, GHOME, HOSTTYPE and Paths

As shipped, WiT is ready to run on the platform you ordered. Before running, you must edit your `~/.cshrc` file to set the `WITHOME`, `GHOME`, and `HOSTTYPE` environment variables. You must also add a path to the WiT executables directory.

1. Edit `~/.cshrc`. For example, if your favorite editor is `vi`, then type:

```
vi ~/.cshrc
```

2. Set the following environment variables by adding the lines below. The `witArch` script is used to set the `HOSTTYPE` variable.

```
setenv WITHOME "/usr/local/wit"
setenv GHOME "$WITHOME/G"
setenv HOSTTYPE "'$WITHOME/bin/witArch'"
```

3. Add the WiT `bin` directory to your shell path based on your host machine type.

```
set path = ($path $WITHOME/bin/$HOSTTYPE)
```

4. Save the file and exit the editor.
5. Update your current C-shell.

```
source ~/.cshrc
```

To run WiT, you must first start a program called `witras` on every machine which may run WiT or a WiT server. The `witras` program runs in the background and must be started under the root user id. You only need to do this once on each machine running WiT including the license host machine. You may then start WiT. Here is the procedure:

1. Start `witras` once under root:

```
su root
witras &
```

2. Now start `wit`:

```
cd $WITHOME/demo
wit
```

The directory from which you started WiT becomes the initial working directory. The working directory can be changed interactively later (see Section 13.8). If you have followed the instructions so far, then your initial working directory will be `$WITHOME/demo`.

2.2 Basics

As with many complex tools, most people prefer to learn by experimentation rather than going through thick manuals. Chapter 3 presents a short tutorial with step-by-step instructions which should be easy to follow and not take too long to go through. However, if you want to skip even that and start experimenting right away, please read this section at least! (If you are not familiar with Motif or OpenLook, whichever you are using, you should also read Section 2.3 first.)

Like all graphical programs, most of your interaction with WiT is done via your mouse. WiT requires a three button mouse to operate. *If you have a two button mouse, you can simulate the middle button by holding down the* **control** *key and using the left button.* Specifically, WiT recognizes these mouse actions for various functions:

Button	**Down**	**Up**	**Drag**
Left	Select. Start area select. Set position.	Done area select.	Move and modify objects
Middle	Add to or remove objects from selected list.	Backup when entering polyline.	
Right	Display pop-up menus.		

With shift key held down:

Button	**Down**	**Up**	**Drag**
Left	Start area zoom.	Done area zoom.	Update area zoom.
Middle	Start pan windows.	Done pan windows.	Pan windows.
Right			

Context sensitive on-line help is available for most objects, such as operator icons, links, Igraphs, buttons and other input gadgets. On SUN workstations, help is invoked with the **HELP** key. On other computers or on keyboards without a **HELP** key, use the **F1** key.

With this knowledge, you are now ready to explore WiT. If you do want to read the manual, plentiful cross-references, marked by the symbol § followed by a section number, have been included wherever appropriate.

2.3 GUI Environment

Before you start experimenting with WiT, it is important that you know the basic techniques to interact with an application that uses the GUI (Graphical User Interface) look-and-feel scheme on your computer. If you are already familiar with it, then you can skip this section.

The following sub-sections describe each GUI which WiT supports. You only have to read about the environment that you are using.

Only the Graphical User Interface (GUI) techniques relevant to WiT are discussed here, so that you can quickly learn the basics and proceed to explore WiT. We recommend that you read the appropriate user's manual for more in-depth information about the GUI you are using.

2.3.1 The Motif Environment

Mouse Buttons and Movement

Motif works primarily with a three-button mouse. Within WiT in general, the left button is used for selecting things and most other actions. The middle button is used for adjusting things, such as augmenting a previous selection set. The right button is used for invoking a menu.

Additional actions are sometimes invoked by pressing a mouse button when the shift or control keys are held down.

When you move the mouse with none of the mouse buttons held down, you are *moving* the mouse. When you move the mouse with any one of the mouse buttons held down, you are *dragging* it.

Panel Buttons

Buttons should be pressed by using the left mouse button. Actions associated with a button are normally executed when a button is *released*.

Menu Buttons

Menu buttons are typically found at the top of a window, on what is commonly called a menu bar. These buttons display a menu when pressed using the *left* mouse button (or *right* button, but this a WiT extension, not a general Motif scheme). You can either *drag* the mouse to the item you want and then release the mouse button there to select it, or release the button and then *move* to the item you want, and click again.

WiT also provides an extension to the standard Motif menu buttons. If you drag a menu button with the middle mouse button, you can detach the menu and place it anywhere on the screen. Subsequent clicks on this menu will not bring it down. Reconnect the menu to the menu button by clicking the left mouse button at the menu button.

Resize Corners

Motif windows are decorated with special corners which may be selected and dragged to resize the window. Resize corners are used in WiT to permit the user to alter the size of graphs, images, the Igraph workspace, etc. If a particular window cannot be resized, then you will not be able to drag its resize corners.

Disabled Items

When an item is not applicable in the current context (such as a delete action when there is nothing to delete), it is disabled so that the user will not accidentally try to execute an invalid action. Disabled items are shown with a gray dithered appearance.

2.3.2 The OpenLook Environment

The Push-pin

One of the most distinct features of an OpenLook application is the use of the push-pin, found on the upper left corner of some pop-up windows. The convention here is that windows that pop-up with the push-pin *out* are intended to be temporary. Normally you make some required selections or enter some data in this window. After the selections are made, appropriate actions are carried out and the window is taken down automatically.

The presence of the push-pin allows you to pin up a window so that even after the required user input is performed, the window will stay up. This is convenient if you know that you will be invoking a particular window quite often.

When you pull out a push-pin from a window, that window will be taken down, along with whatever action (possibly none) is appropriate.

If a window comes up with the push-pin out, but you want to dismiss it without entering any data, you can push the pin in and then pull it back out.

Mouse Buttons and Movement

OpenLook works primarily with a three-button mouse. In general, the left button (called the *select* button) , is used for selecting things and most other actions. The middle button (called the *adjust* button) is used for adjusting things, such as augmenting a previous selection set. The right button (called the *menu* button) , is used for invoking a menu.

Additional actions are sometimes invoked by pressing a mouse button when the shift or control keys are held down.

When you move the mouse with none of the mouse buttons held down, you are *moving* the mouse. When you move the mouse with any one of the mouse buttons held down, you are *dragging* it.

Panel Buttons

Buttons should be pressed by using the left mouse button. Actions associated with a button are normally executed when a button is *released*. In a pop-up window with a push-pin, clicking a button normally signifies the end of data entry. Buttons that appear in a pop-up window often imply this, such as **Apply** and **Cancel**.

Menu Buttons

Menu buttons are shown with a triangle (down arrow) on the right edge of the button. These buttons bring up a menu when pressed using the *right* mouse button. You should drag the mouse to the item you want and then release the mouse button there to select it. A short-cut to select the first item on the menu is to press the *left* mouse button at the menu button.

Resize Corners

Most OpenLook windows are decorated with special corners which may be selected and dragged to resize the window. Resize corners are used in WiT to permit the user to alter the size of graphs, images, the Igraph workspace, etc.

Disabled Items

When an item is not applicable in the current context (such as a delete action when there is nothing to delete), it is disabled so that the user will not accidentally try to execute an invalid action. Disabled items are shown with a gray dithered appearance.

2.3.3 On-line Help

On-line help is available by hitting the **Help** or **F1** key on the keyboard with the mouse cursor pointing to the object for which help is required. Objects can WiT operators, or other WiT objects such as links. (If you have *click-to-type* turned on for your window manager, then you must make sure that the WiT window is selected before attempting use of the **Help** key.)

2.4 Floating License

WiT is copy protected by a floating license mechanism. When you purchased WiT, the machine ID that you supplied is used to set up a password that allows the WiT license manager to run only on that particular machine. We will call this machine the license host machine. The license manager knows about how many copies (specified when you purchased WiT) you are authorized to run. You can run this number of copies of the WiT *UI* on any machine on the same local area network as the

license host machine. There is however no limit to the number of WiT servers that you can connect to the UI.

If there is more than one WiT user in your environment, then there is concern to properly distribute user files so that each user can work both independently *and* cooperatively with everybody else. This is particularly complex because WiT is an open-ended system. Users can add their own operators, change icons, create new libraries, add new object types, and even develop new servers. Some of these you may want to share with other users, others you may want to keep private. Since this is a fairly advanced topic, its discussion is deferred to Section 13.6.2. For now, we will just assume that you are the only user.

Chapter 3

Quick Tour

The purpose of this quick tour is to introduce you to WiT as quickly as possible. The emphasis is on showing you how *easy* it is to execute operators and use WiT features while avoiding unnecessary details. Complete details will be given in subsequent chapters.

WiT should look like Figure 3.1 when it first comes up.

The main WiT window consists of three areas: the menu bar, the tool bar, and a large workspace on the bottom. In addition, there is a detached status window for messages.

Operators are selected from the **Operators** menu and placed in the workspace. They are then connected with links to form a functional Igraph. Execution of an Igraph is controlled from the **Run** menu.

3.1 Exercise 1: Displaying an Image

Let us start our tour with a simple task: read an image from a file and display it. Many people like to describe tasks and algorithms with block diagrams, because they are visual and easier to understand than a textual description. It would be nice if the same block diagram can actually be used for the *execution* of the algorithm. This is precisely what WiT lets you do: it allows you to specify an algorithm with a graph (block diagram), called an *Igraph* (for imaging graph). Igraphs are made up of icons (blocks) connected with links. Icons represent operators, e.g. read an image, perform a convolution, etc. Links specify paths along which data travels from one operator to the next. The order in which operators are executed is determined by the flow of data. Figure 3.2 shows the Igraph that would do our job.

3.1.1 Building the Display Image Igraph

Now let us build this Igraph in WiT. You will see that not only is the Igraph a concise and visually pleasing representation of the algorithm, but that it is also much easier to create and run than an ordinary computer program.

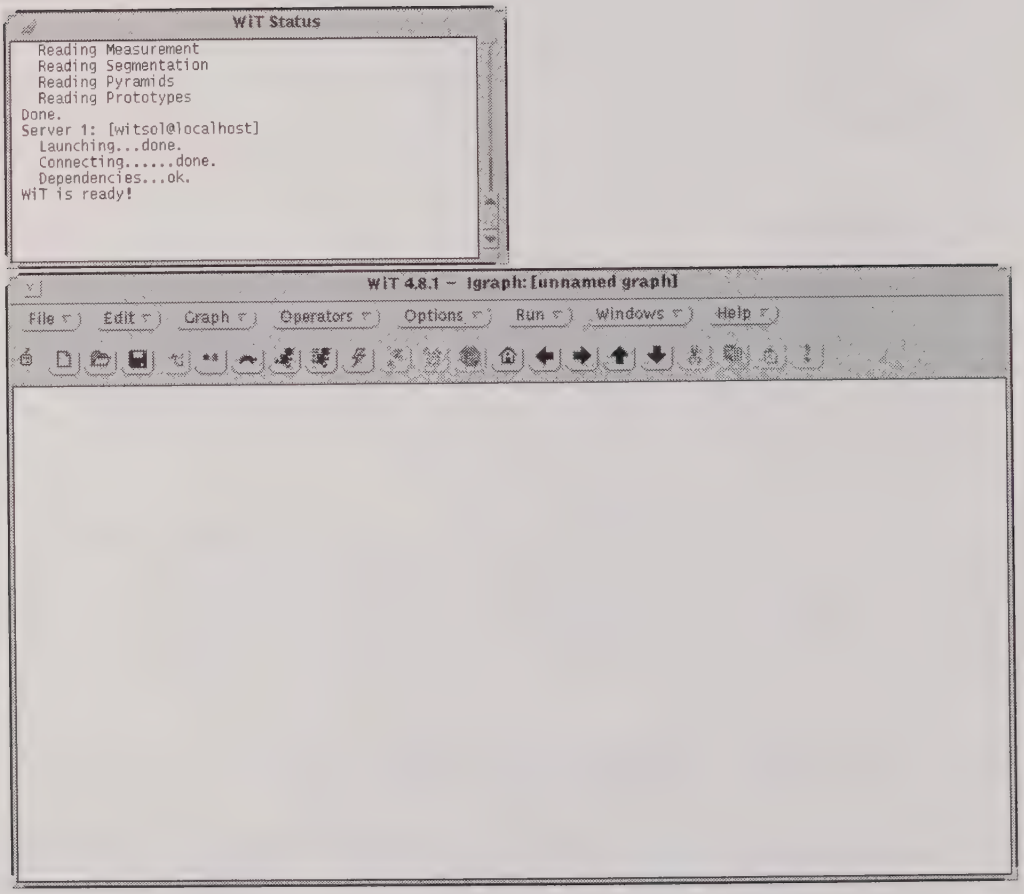

Figure 3.1: WiT on Start Up

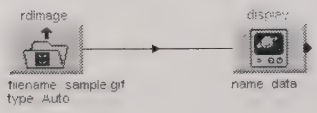

Figure 3.2: Igraph for Reading and Displaying an Image

Creating an Operator to Read an Image

First we need an operator that can read an image. Pull down the **Operators** menu. You will see a list of library names, grouped according to function. Select the **Read/Write** library. A new window comes up (Figure 3.3). These are the operators which WiT supports for reading and writing data files.

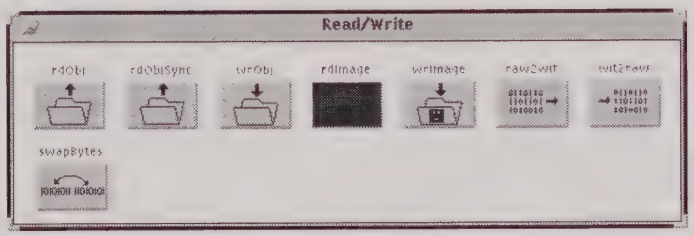

Figure 3.3: Read/Write Library Panel

Click (press and release) the **rdImage** icon with the left mouse button. The main window comes to the front. Your mouse cursor, with a copy of the **rdImage** icon attached, is automatically placed at the center of the workspace. As you move your mouse now, the **rdImage** icon follows. Move to somewhere to the left of the workspace, and click the left mouse button. This creates an instance of the **rdImage** operator, and a properties panel (Figure 3.4) pops up.

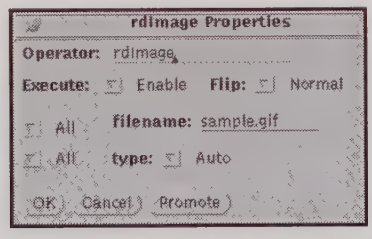

Figure 3.4: RdImage Properties Panel

Each operator has its own unique properties panel. For example, **rdImage** has a **Type** property which lets you define what image type you want to read, such as GIF, TIFF, or BMP. If this is the first time you run WiT, all the default parameters will be set up for this tutorial, so just click the **OK** button. The properties panel goes down. Notice the properties are shown below the operator icon.

Creating an Operator to Display an Image

Next we need an operator to display the image. Again pull down the **Operators** menu. This time choose the **Interactive** library. The **Interactive** library window (Figure 3.5) appears. Choose

display and follow the same procedures to place the icon on the Igraph. Put it somewhere to the right of **rdImage**. When the properties panel come up, again accept all the default parameters by simply clicking **OK**.

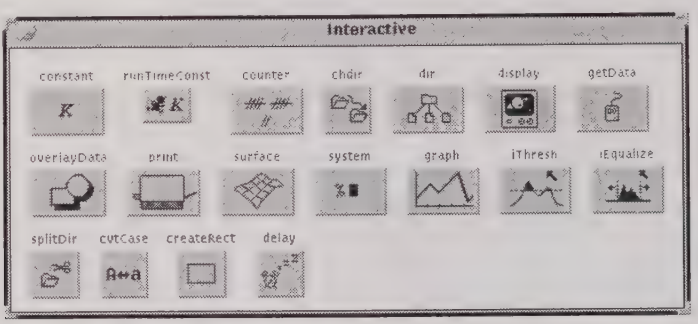

Figure 3.5: Interactive Library

Adjusting Icon Positions on an Igraph

If you are not happy with the positions of **rdImage** or **display** on the Igraph, point your mouse at the operator icon you wish to move, then press and *hold* the left mouse button. Now as you *drag* your mouse, the operator follows. The operator is placed wherever you release the mouse button.

Creating a Link to Connect RdImage and Display

At this point, you should have something like Figure 3.6 on the workspace. Notice the black arrow head on the right of **rdImage**, and the black arrow tail on the left of **display**. The arrow head symbolizes an output port: data going out of the operator. The arrow tail symbolizes an input port: data coming into the operator. Input ports usually appear on the left side of operator icons, output ports on the right.

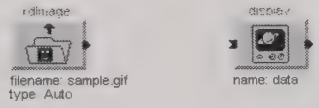

Figure 3.6: Operators

We now need to connect the two icons together with the proper data-flow. Point at the output port of **rdImage** and *double click* the left mouse button. Now when you move your mouse, a rubber

band wire is created between the port and the cursor. Move the cursor to the input port of **display** and double click the left mouse button. The two ports disappear. A link with an arrow indicating the direction of data-flow connects the output of **rdImage** to the input of **display**. You should now have the Igraph shown previously in Figure 3.2.

3.1.2 Running the Display Image Igraph

Click the on the tool bar. The status indicator on the left of the tool bar turns from the static icon (a mouse, symbolizes editing) to an animated icon. **RdImage** turns green for about a second, and a message comes up on the WiT status window stating that **RdImage** is now executing. When **RdImage** is finished, a red dot (called a *token*) travels from the output of **rdImage** to the input of **display** along the link. When the token arrives at **display**, **display** turns green for a second, and a new window pops up, with the image (**sample**) read by **rdImage** shown. WiT reports on the status window that the Igraph execution is complete, and the status indicator on the tool bar changes back to .

3.2 Exercise 2: Studying Images

An image is initially shown at its original size when it first comes up (if the image is larger than the screen, it is scaled to fit), but you can resize it using the resize corners.

A number of tools are available for studying an image while it is displayed. Press and hold the *right* mouse button inside the **sample** image. A pop-up menu appears. Select **Properties**. An image properties window comes up (Figure 10.18). Now try dragging (press and hold) the mouse with the left mouse button down inside the image window. You will see the pixel position and value displayed interactively on the image properties panel. Next check the **X** and **Y** boxes for **Show profile**. Two more windows come up, one for a plot of all pixel values along the x-axis (horizontal line), the other of pixels along the y-axis (vertical line). Again, as you drag your mouse in the image window, the profile plots are updated interactively.

On the right of the image properties panel, there is a grayscale ramp with three arrows. (§10.18.1) You can control the contrast of the image by dragging the arrows. For example, press and hold the left mouse button at the top arrow, and drag the arrow down to around 128, for example. The image becomes brighter. Dragging the middle arrow shifts all three arrows without changing the separation between the arrows. Experiment with all three arrows, and observe the effects.

3.3 Exercise 3: Managing Igraphs

To save the Igraph you have just created, click the tool bar button. A file dialog comes up. Enter an appropriate name for the Igraph, say 'mygraph', then click the **OK** button.

To clear the current Igraph, click the tool bar button. The workspace is cleared, and all pop-up data windows (the **sample** image window and its properties) are removed.

To load an Igraph, click the tool bar button. A file dialog comes up. All the Igraphs described in this section have already been pre-built and tested. They are stored in the `demo` directory, named `egXXX`, where `XXX` is the example number where the Igraph is first referred to. So to see what the pre-built Example One Igraph looks like, enter the name `eg1` for the file name, and click the **OK** button. The current Igraph is cleared, and the new Igraph is then displayed in the workspace.

3.4 Exercise 4: Performing a Linear Series of Operations

In our next exercise, we will read a GIF image, convert it to its negative, rotate it 30 degrees, then display the result as a 3-D plot. Figure 3.7 shows the Igraph to do that.

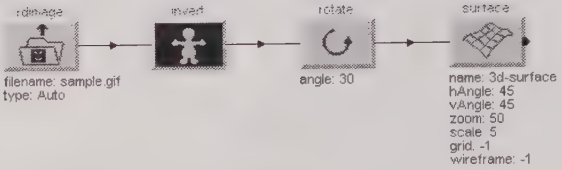

Figure 3.7: A Series of Operators

Now let us build and run this Igraph. If you do not wish to go through the editing process, load the pre-built Igraph `eg4`, and skip the following building instructions.

To build this Igraph, first clear the workspace: click the tool bar button. Create an instance of **rdImage**. Put it somewhere on the left of the workspace. We will use the same **sample** image, so accept all the default parameters by clicking the **OK** button.

Next create an instance of **invert**. Instead of using the graphical libraries (**invert** is in the **Operators/Point** library), this time let us use the text based **Operator Selector**: select **Selector** from the **Operators** menu. The **Operator Selector** panel comes up (Figure 10.7).

The **Operator Selector** is an alternate way to invoke operators. It is not as graphical as the functional library groups, but it allows you to browse through all operators without having to worry about which library they belong to. To make an instance of **invert**, type the word `invert` at the **Name** entry on the **Operator Selector**. You will see the list of operators interactively updated as you type each character. When you can see 'invert' in the operator list, stop typing, and click the left mouse button at 'invert' in the operator list. It is now highlighted. Click the **OK** button. Now you are in exactly the same state as if you had selected **invert** from its functional group: you are ready to place the operator on the Igraph. Place it to the right of **rdImage**. **Invert** has no parameters, so just click **OK**. Connect the output of **rdImage** to the input of **invert** by creating a link.

Create an instance of **rotate** (**Point** library). Set the **angle** parameter to 30. Connect the output of **invert** to **rotate**.

Create an instance of **surface** (**Interactive** library). Use all the default properties. Connect the output of **rotate** to **surface**.

You are now ready to run this Igraph. Click the button. The operators will each turn green for a second, in the order specified by the links. When the token reaches **surface**, a window pops up with a 3-D plot of the image. The pixel intensity is plotted on the Z-axis. Details on the **surface** operator can be found in Section 9.2.

3.5 Exercise 5: WiT Data Objects

Data objects in WiT are represented internally by a special format. This format allows WiT operators to check whether the object it receives is the correct type. For example, if an operator expects an image as input, and you send it an integer, the operator can issue a warning and refuse to process the data. Some operators can deal with a variety of input types, doing different things depending on the data type. This data format is preserved when a WiT data object is saved to a file, so that when a data object is read back, WiT will know what data type it is. It also means that only one operator (**rdObj**) is required to read a data object from a file, and only one operator (**wrObj**) is required to write a data object to a file.

In this exercise, we will explore how WiT handles different data types with the Igraph shown in Figure 3.8.

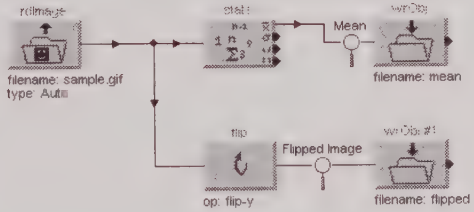

Figure 3.8: Igraph for Demonstrating Use of Data Objects

This Igraph introduces some new Igraph features:

- Some links (between **stats** and **wrObj**, and between **rdImage** and **flip**) are not simple straight lines.
- The link from **rdImage** *branches* off into two links,
- There are two instances of the same operator **wrObj**, and one of them has a **#1** attached to its name.
- There are magnifying glasses on the links between **stat** and **wrObj** and between **flip** and **wrObj #1**.

Because of these new features, it is worthwhile for you to go through the editing process for this Igraph, although a pre-built version does exist (`eg5`).

First create one instance of **rdImage**, **stat**, and **flip**, and place them at the appropriate positions. To create the link branches, first connect the output of **rdImage** to the input of **stat** as usual. Now point the mouse at the middle of this link and double-click the left mouse button. A circular dot appears at the position where you clicked your mouse. This is a newly created link *junction*. As you move your mouse, a rubber band link exists between the mouse cursor and the junction. Click the left mouse button once to create a link corner, then point at the input port of **flip** and double-click the left mouse button. The link branches are now properly created.

Next make an instance of **wrObj**. Set the **filename** parameter to **mean**. Connect the first output (the mean) of **stat** to its input. Leave the other outputs of **stat** unconnected.

Make another instance of **wrObj**. Set the **filename** parameter to **flipped**. Place it to the right of **flip**. Notice the name is automatically set to **wrObj #1**. WiT does this because all objects (operators and links) in an Igraph must have unique names. On the properties panel, you can change the name of the operator to almost anything you like (see Section 6.6.8 for restrictions), provided it does not conflict with any other objects in the Igraph. If the name you type is already in use, WiT will append a number to it. Connect the output of **flip** to **wrObj #1**.

The only remaining components now are the magnifying glasses (the magnifying glass is part of the WiT logo!). The magnifying glass (or probe for short) allows you to inspect a data token without having to invoke the **display** operator. A probe can be placed anywhere on a link. When a data token travels on that link, a pop-up window will be created to display the contents of that token. To create a probe on the link between **stat** and **wrObj**, click the *right* mouse button at the position you want the probe to be placed. A properties panel for the link comes up. Click **Yes** for **probe**. Notice the link has a name, which starts with an underscore ('_'). Igraph object names that start with an underscore are not displayed. This feature is used to avoid displaying meaningless names which can clutter the appearance of an Igraph, particularly so for links. However, when we place a probe on a link, the name of the link will be used as the name of the pop-up data window later, so it is helpful to give the link a meaningful name. Name the link **Mean**.

Repeat the process for the link between **flip** and **wrObj**. Name the link **Flipped Image**.

Now run this Igraph. As the token from **rdImage** reaches the link junction, it splits into two and the new tokens travel simultaneously to the input ports of **stat** and **flip**. Later you will find that this branching action enables WiT to make use of multiple computers to speed up execution of Igraphs. As the tokens travel under the magnifying glasses, two pop-up windows will appear (Figure 3.9). One is named **Flipped Image**, and contains an flipped image of **sample**. The other is named **Mean**, but this one contains a single number, which is the mean of all the pixel values in the **sample** image. This number is also a WiT data object. A WiT data object can be simple, such as a single integer or character string, or it may be complex, consisting of nested data structures and arrays. Images are in fact just a particular type of data object. All data objects other than images are displayed as formatted text.

These two data objects are saved to files by **wrObj**. Notice you do not need to use different operators for different object types. **WrObj** handles *all* WiT data types.

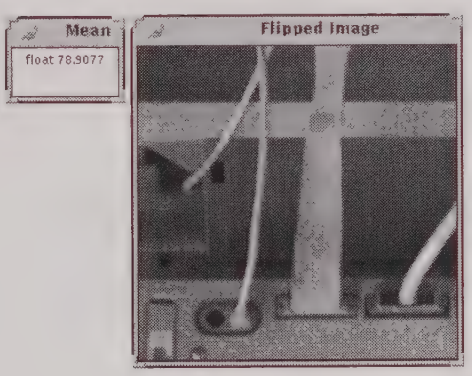

Figure 3.9: Data Objects

3.6 Exercise 6: Working with Frame Grabbers

If you have purchased a frame grabber server with WiT, you can try this exercise. Otherwise, skip this section.

It does not matter which frame grabber you are using, as long as you have purchased the correct WiT server for the frame grabber. This is in fact a very important WiT feature: similar to high-level programming languages, the Igraphs you design are independent of the hardware you are using. This means you can change hardware without modifying your Igraphs. Even if your frame grabber has on-board processing abilities, most of these functions have been mapped to portable high-level WiT operators.

Again analogous to programming languages, WiT does offer low-level hardware dependent operators for each supported frame grabber for uncomprised flexibility. This is similar to the use of assembly instructions within a high-level language program. Its use is not recommended, but sometimes it is the only way to achieve certain program behavior. Just as in programming languages, you can encapsulate use of low-level WiT operators in functions (hierarchical WiT operators) so that it is easier to replace hardware dependent operations when the hardware is changed. Refer to the appropriate WiT server documentation for descriptions of low-level operators.

If you have followed the installation procedures for the server you purchased, WiT would have come up with everything correctly configured.

In this exercise, we will simply grab an image and display it (Figure 3.10).

Build this Igraph yourself or load `eg6`. Run it and you will see an image grabbed from your camera! What can be simpler? You can insert some processing between **acquire** and **display**. Depending on the capabilities of your frame grabber, some or all WiT processing operators may run on the hardware.

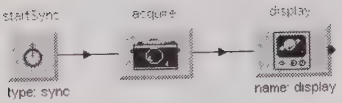

Figure 3.10: Igraph for Grabbing an Image

3.7 Exercise 7: Executing Operators in Demand Mode

Although Igraphs are extremely powerful and easy to use, there are times when you just want to apply one or two operators to some data objects. For example, you may want to read an image and transpose it, then save it back. Having to build a tiny Igraph and then run it is inconvenient. Especially when you know you will never use the Igraph again. So WiT provides a demand-driven mode of execution. In this mode, operators that you select are executed and the results shown immediately. Demand-driven mode is useful in exploring and understanding the rich set of imaging operations WiT offers, as well as debugging any new operators that you have developed yourself.

You can switch to demand-driven mode by clicking . To go back to data-driven mode, click .

The menu bar and tool bar remains the same in demand-driven mode, although some items may be disabled. The workspace, however, looks very different (Figure 3.11). It consists of five blue rectangular areas. These are meant to resemble playing cards (§7.2). They represent storage places for images (and data objects in general).

Operators are invoked just as in data-driven mode. If an operator requires inputs, they are retrieved form the card stack and therefore must exist on the cards before the operator is invoked. After execution, any outputs an operator produces are pushed on the card stack.

The cards function as a last-in first-out stack: output data objects are pushed on the left of the stack as they are produced (i.e. the last output object will end up as the left-most card), and input objects are read beginning from the left of the stack (i.e. the first input object is fetched from the left-most card).

In this exercise, we will compute the average pixel value of a region within an image in demand-driven mode.

Invoke **rdImage** just like when in data-driven mode. A panel like Figure 3.12 appears. Notice that some graph related parameters, such as **Flip** and **Disable**, are not present. Set **filename** to **sample**. When you click the **OK** button, **rdImage** is executed immediately (instead of being placed on an Igraph), and the image appears on the first card in the workspace. All objects are shown with a fixed standard size on a card. To look at the **sample** image at the original resolution, click the left mouse button on the first card. The card will momentarily move to a lower position and then fall back to its original position (more about this behavior later), and a larger image appears in a detached window.

Notice the name **rdImage** shown on the title bar of the detached window. It is inherited from the name of the operator that produced this image. So if you want to read another image and have it

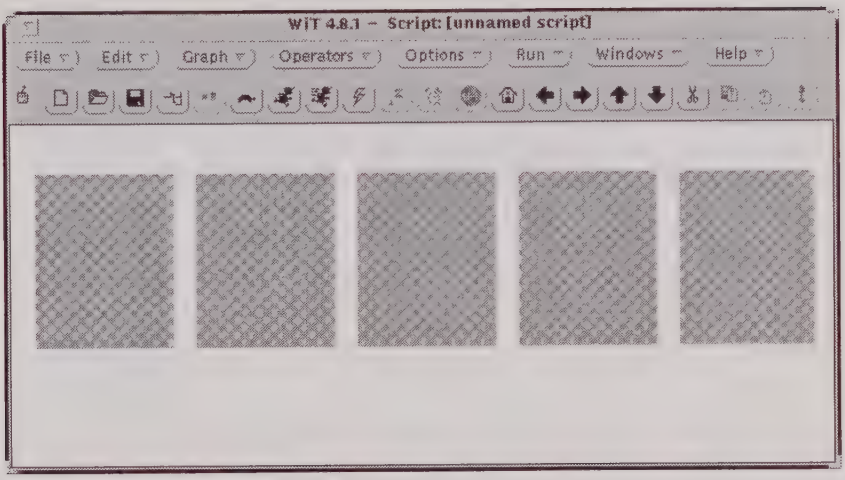

Figure 3.11: WiT in Demand-Driven Mode

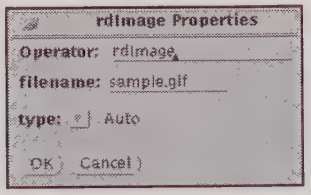

Figure 3.12: RdImage Operator Properties in Demand-Driven Mode

displayed in another detached window, you need to change the operator instance name before you click **OK**.

Invoke the **getData** operator. Set the **type** to **rectangle**, and click **OK**. The **sample** image appears in a separate window, with the **getData** panel which we have already seen previously. Enter a rectangle for the region you are interested in. Then click **OK**. A new card appears. Because the output of **getData** is a vector of graphic objects, this new card does not contain an image. It simply says `Graphic[ ]` on it. It means this card contains a vector (indicated by the brackets) of objects of type **Graphic**. You can look at the contents of this object by clicking on the card, just as you did to display the image previously.

Now we would like to execute the **roiStat** operator. However, **roiStat** requires a *single* **graphic** object, not a vector. Even though the **getData** output is a vector of just one element, it is still a vector. To extract the single element from the vector, invoke the **getElem** operator. Set **method** to **first**, and click **OK**. A new card is now produced. Notice its type is **Graphic** (without the brackets),

meaning it is a single **Graphic** object (Figure 3.13).

Figure 3.13: After Executing Operator getElem

Now we are finally ready to execute **statRoi**. It requires an image on its first input and a graphic object on its second input. WiT searches the card stack from left to right for an object of the correct type for each input required for an operator. Even though the order of the image (the **rdObj** card) and the graphic (the **getElem** card) are in the wrong order, WiT is able to feed the correct inputs to **statRoi**. The four outputs of **statRoi** goes to four new cards. Since each output is a simple floating-point number, their values are shown directly on the cards.

3.8 Exercise 8: Manipulating the Demand Mode Stack

Now suppose we want to save the **mean** value from the previous exercise to a file using **wrObj**. However, if we execute **wrObj** now, it will save the first (left-most) card instead of the one we want (**mean**, third from the left).

To save the **mean** card, we need to make it the left-most card. Point the cursor at the card you want and press and *hold* the left mouse button. The card will move to a lower position, and an arrow will appear to indicate the insertion point that the card will go to (Figure 3.14). With the left mouse button held down, move the cursor to the left of the left-most card, then release the button. Notice all other cards are pushed to the right by **mean**. Also note that if you release the button while the arrow is pointing to the original position of the card (this is the same as clicking the left mouse button on the card), WiT will take that as a request to display the object in a pop-up window.

Try moving different cards around. You can use the edit menu on the menu bar or the tool bar edit buttons to copy and paste cards. Only the left-most card can be copied, and a card is always pasted at the left-most position. The same workspace menu is also available in demand-driven mode.

3.9 Exercise 9: Advanced Igraph Editing

Both operators and links are considered *Igraph objects*. Many editing techniques apply to both operators and links. You have already tried the technique for bringing up the properties panel.

Figure 3.14: Moving a Card in Demand-Driven Mode

In this exercise, you will be introduced to several other useful techniques. First switch back to data-driven mode (click), and clear the workspace.

3.9.1 Selecting Igraph Objects

Create some operators and links on the Igraph. To select an Igraph object, click the left mouse button while pointing at it. Notice the object turns red. Press and hold the left mouse button at a blank location on the Igraph, then drag the mouse. The rubber band box performs an area select. To add an object to the selected list, click the middle mouse button. To remove an object from the selected list, click the middle mouse button *again*.

3.9.2 Moving Objects

Select a few objects (both operators and links). Point the mouse at one of the selected objects, then press and hold the left mouse button. Drag the mouse. Notice all the selected objects move with the mouse. Release the mouse button. The selected objects move to the new location.

Move the mouse to a blank location, then press and hold the right mouse button. A pop-up menu comes up. Select **Undo**. The objects move back to their original position. You can undo any editing action, such as copy, paste, or even link modification (see following sections).

If you move a *single* operator which has links attached to it, then all the links rubber band to follow the operator. This way you can shift operators around without having to reconnect its links.

3.9.3 Cut, Copy, Paste, and Duplicate

Select a few objects. Select **Cut** from the workspace pop-up menu. The selected objects are deleted. Bring up the pop-up menu again, and select **Undo**. The previously deleted objects come back. Select **Duplicate** from the pop-up menu. A copy of the selected objects now follows the mouse. Click the left mouse button. The new set of objects is placed at where you clicked the mouse.

Select **Copy** from workspace menu. Click . Select **Paste** from workspace menu. A copy of the selected objects now follows the mouse. Click left mouse button to place the objects.

3.9.4 Editing Links

Make some links with corners on the Igraph. Deselect everything by clicking the left mouse button at a blank location. Now point at the a link corner, then drag the mouse with the left button down. The two line segments connected to the link rubber band to follow the mouse. Release the mouse button. The link shape is modified. Notice that the link must *not* be selected when you modify its shape. If the link is selected, then clicking the left mouse button at it will start *moving* the entire link.

3.9.5 Icon Appearance

From the operator properties panel, you can flip an operator, control how each parameter is displayed, disable it, or even bypass it. A disabled operator (or link) behaves as though it does not exist on the Igraph. A bypassed operator behaves as though there are links which connect its inputs directly to its outputs. Naturally, bypass is only possible with operators which have the same number of input and output ports *and* the ports support the same data types. Examples of operators which can be bypassed are **lopass2d**, **invert**, and **median**, just to name a few.

3.9.6 Panning and Zooming

If your Igraph gets too big, you may want to zoom out (make the objects smaller) so that you can fit more operators on the screen.

Do an area select, but with the shift key held down. If you move the mouse to the right and down while doing the area select, you will zoom in (objects become bigger) the Igraph. The mouse cursor changes to . If you move the mouse to the left and up, you will zoom out (objects become smaller) the Igraph. The mouse cursor changes to .

To pan the Igraph (especially useful after you have zoomed in), drag with the middle mouse button down and the shift key held down. To return to the original size, select **Normal size** from the workspace menu. To home the Igraph, click .

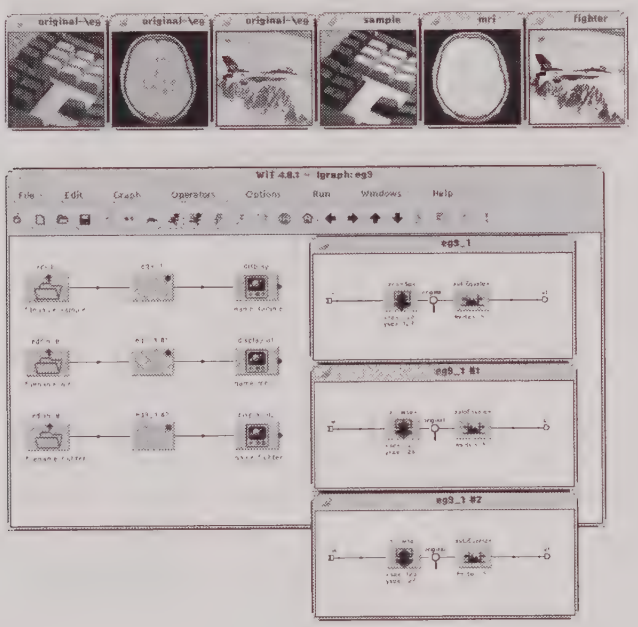

Figure 3.15: Hierarchical Operator Usage

3.10 Exercise 10: Hierarchical Operators and Icon Editor

Similar to procedures or subroutines in a conventional computer program, it is often helpful to create a new WiT operator from an Igraph which performs a useful set of operations. These operators are called *hierarchical* operators. The Igraph which it represents is called a sub-graph. In this exercise, we will make a hierarchical operator which scales an image to 128×128, enhance the contrast by equalizing it, and returns that as the output. Then we will make multiple instance of this new operator to process several images within one Igraph. Figure 3.15 shows the main Igraph and the sub-graph that we will build.

3.10.1 Creating the Sub-Graph

We will first make the sub-graph. Clear the workspace (click). Create one instance of **zoomSize**. Set both the **xsize** and **ysize** parameters to 128.

Create one instance of **autoEqualize** (**Point** library). Leave the **fraction** parameter at the default value of 5. Connect the output of **zoomSize** to the input of **autoEqualize**. Enable probe on this link. Name the link (probe) **original**.

Create one instance of **in** (**Dataflow** library). Notice that the symbol of the **in** operator looks like an ordinary operator input port, but it is hollow and slightly larger. This symbol is *not* an operator port. On the right is a solid output port: this *is* an operator port. It is the output of **in**. Connect this port to the input of **zoomSize**. Later, when we make a hierarchical operator out of this Igraph, the **in** operator will correspond to a real operator input port on the hierarchical operator.

Create one instance of **out** (**Dataflow** library). Like the **in** operator, the **out** operator has a hollow output port shaped symbol, and a real input port. Connect the output of **lopass2d** to **out**. **Out** operators will correspond to a real operator output port on a hierarchical operator.

Save the Igraph as `myOp` (a pre-built version of this Igraph exists as `eg9_1`).

3.10.2 Making the Sub-Graph an Operator

The Igraph which we have made cannot be run as it is, since it is lacking a source for the input data. We want to use it like

Select the **Graph/Make Operator...** menu item. The **Make Operator** panel comes up (Figure 6.3). Notice the name of the operator is set to `myOp`, the name of the Igraph. the number of inputs and outputs correspond to the number of **in** and **out** operators in the Igraph. The icon width and height values default to a reasonable size such that all ports will fit comfortably on the icon. You can modify the width and height if you want, but for this exercise, leave them as they are.

Below to the width and height fields is a choice list for which library you want to put this new operator in. Select the **Prototype** library. Then click **OK**.

Bring up the **Proto** library (select **Operators/Proto** menu item). You will see an icon named **myOp** in the panel! You are now ready to use this new operator. The red dot on the upper right corner indicates that this is a hierarchical operator.

3.10.3 Using the Hierarchical Operator

Clear the workspace (click). Create three instances of **rdObj** (you may want to use the **Duplicate** workspace menu item). Set the **filename** parameter of the first instance to `sample`, the second to `mri`, and the third to `fighter`.

Create three instances of **myOp**. Connect the output of each of the **rdObj** operators to an instance of **myOp**.

Create three instances of **display**. Set the **name** parameter of the first instance to `sample`, the second to `mri`, and the third to `fighter`. Connect the output of each of the **myOp** operators to an instance of **display**.

Now bring up the properties panel for one of the **myOp** operators. Notice that it has a button labeled **Expand**. Click this button, and you will see the Igraph that **myOp** represents come up in a pop-up window.

Run the Igraph while the **myOp** sub-graph is visible. You will see that visual feedback (operators

turning green, token travel, etc.) applies to sub-graphs also.

3.10.4 Modifying Sub-Graph Behavior

Since a hierarchical operator (and hence its associated sub-graph) can be instantiated many times in one Igraph, and the same operator can be incorporated into many Igraphs, naturally you should not be allowed to modify a sub-graph. However, often you may want to put a probe on a sub-graph link, or disable an operator or link within it, without affecting other instances of the sub-graph. You *are* allowed to change operator and link parameters (including probes) on a sub-graph. For example, you can add a probe to the link between **zoomSize** and **lopass2d** in the sub-graph of one of the instances of **myOp**. If you open up another instance of **myOp**, you will see that it is not affected by the changes you made. You can even add links to a sub-graph, but you are *not* allowed to add operators. You cannot delete objects from a sub-graph.

Sometimes you want to access some parameters of an operator in a sub-graph without having to expand the corresponding hierarchical operator. For example, you may want to promote the **xsize** and **ysize** parameters of **zoomSize** to **myOp**. Parameter promotion is discussed in Section 6.7.3.

3.11 Learning More

In the `demo` directory, there is a large collection of interesting Igraphs besides the ones discussed in the preceding exercises. Load and study it each one. Hit the **Help** key at a blank area on the Igraph to get information about what the Igraph is supposed to demonstrate. Run the Igraph and observe the results.

After you have tried all the demo Igraphs, you should have a fairly good feeling of what WiT can offer you. Most likely you will want to design some Igraphs of your own, and try out the built-in operators that come with WiT. The rest of this *User's Manual* describes in complete detail all the features of the WiT GUI. For help on operators, bring up the help window for the operator (§4.4). If you know what processing effect you want but do not know what operator will produce the effect, try doing a keyword search (§10.6). If you are comfortable with writing programs, you can proceed to the *Programmer's Guide*. There you will learn how to add your own operators to WiT, and communicate seamlessly with the built-in operators or other operators that you have designed. You may even want to create whole new libraries, and use WiT for something totally unrelated to image processing!

Chapter 4

Operators

Operators in WiT are analogous to instructions or functions in a programming language. The connectivity of operators is analogous to the sequence of instructions within a conventional program. Instructions can only be arranged in a linear fashion (at least visually). But operators in a WiT Igraph can be arranged in different topologies, which make Igraphs much better for representing parallel algorithms.

4.1 Invoking Operators

In general, there are two ways to invoke an operator:

- Graphical operator panels (§4.1.1).
- The text-oriented Operator Selector (§10.5).

An operator library can be specifically created so that it does not appear as a graphical panel. This is sometimes done to libraries which are very large, or where no icons have been designed. For operators in those libraries, the only way to invoke them is through the Operator Selector.

4.1.1 Graphical Operator Library Panels

Operator library panels are typically grouped by function. For example, the **Read/Write** library contains operators for reading from and writing to files, and the **Point** library contains operators which process images on a point-by-point basis (no interaction among points). Figure 4.1 shows an example of an operator library panel.

Invoking an Operator from a Library Panel

The steps involved in invoking an operator from a library panel are slightly different in data-driven and demand-driven modes.

Figure 4.1: Point Library Panel

Data-Driven Mode:

1. Bring up the appropriate library panel from the **Operators** menu.
2. Locate the operator you want within the library panel.
3. Click (press and release) the left mouse button at the operator. The main window comes to the front automatically. The mouse, with a copy of the selected operator icon attached, is automatically placed at the center of the workspace.
4. Move the mouse to where you want the operator to be placed. The operator icon will follow your mouse movements.
5. Click the left mouse button to place it on the Igraph. You can only place operators on the main Igraph, not sub-graphs (§6.7).
6. The operator properties panel comes up. Make any desired changes to the parameters, then click the **OK** button. If you click the **Cancel** button instead, changes you made to the parameters will be discarded, but the operator is still retained on the Igraph.
7. You are now ready to select the next operator.

Demand-Driven Mode:

1. Bring up the appropriate library panel from the **Operators** menu.
2. Locate the operator you want within the library panel.
3. Click (press and release) the left mouse button at the operator. The main window comes to the front automatically.

4. The operator properties panel comes up.

5. Make any desired changes to the parameters, then click the **OK** button. The operator is executed and its results, if any, are placed on the workspace cards. If you click the **Cancel** instead, the operator will not be executed.

6. Notice the status indicator changes to . You must wait for the operator to finish executing (the status indicator changing to) before invoking the next operator.

7. You are now ready to select the next operator.

4.1.2 The Operator Selector

Often you know the name of the operator you wish to execute, but you are not sure which library it is in. Sometimes you only remember part of the name of the operator you want. Going through the graphical operator library panels to locate such an operator can be a frustrating task. Also, some large libraries do not have library panels. The Operator Selector panel (§10.5) is provided to address these requirements.

To invoke an operator from the Operator Selector:

1. Bring up the Operator Selector window from the **Operators** menu. It is the first item on the menu.

2. Type the name of the operator in the **Name** field. See below (§4.1.2) for different ways to enter the name.

3. Hit the **OK** button. From this point on, WiT behaves the same way as if you have selected the operator from the graphical operator panel (§4.1.1).

There are a few more features in the Operator Selector which helps the selection of a desired operator:

- The **Filter** choices allow you to display only primitive and hierarchical operators, or both.
- Double-clicking an entry in the operator list is a short cut for choosing and invoking an operator without hitting the **OK** button.
- Use the **Search...** button to locate an operator by using keywords (§10.6).

Entering an Operator Name

To enter an operator name in the Operator Selector, do one of the following:

- Type the name of the operator on the **Name** field.

- Type *part* of the operator name on the **Name** field. You can use the '*' wild card character. As you type, the operator list (the one on the right hand side) is updated to show the list of operators which match the characters typed so far. You can scroll this list at any time to locate the complete name of the operator you want. Click the left mouse button at the operator you want. It is copied to the **Name** field and the library that it belongs to is highlighted on the library list on the left.

- Click a library name (or **all**) on the library list. Operators from that library will be listed on the operator list. Click the left mouse button at the operator you want.

4.2 Operator Icons

An operator icon can be any size. It can have any number of input ports, outputs ports, and parameters. Normally, an operator is shown as an icon with the operator name at the top, parameters at the bottom, input ports on the left, and output ports on the right. The icon may or may not have a 3-D look.

WiT collects information about an operator icon from two places:

- A definition entry in a `def` file, where information about port positions are retrieved.

- An icon bitmap file for the graphics inside the icon.

These are explained in more detail below. You may also want to refer to Section 13.7 for a description on WiT directory structures.

4.2.1 Operator Definitions

Operator definition files are located in the library search paths defined in the WiT configuration file (§13.3). Take the **wrObj** operator as an example. In the default configuration file, `libpath` is set to `$WITHOME/lib`. Because **wrObj** belongs to the **io** library (the short name (§13.4.3) for the **Read/Write** library,, there should be a file called `$WITHOME\lib\def\io`. In it is an entry for **wrObj** (among others):

```
operator
wrObj {
  input object (0,20);
  param file filename = "outfile";
  output sync (60,20);
}
```

This entry contains information about the ports, parameters, whether WiT should draw a 3-D frame around it, etc. The syntax description is explained in the *Programmer's Manual*.

Unless you are writing you own primitive operators (those written in C), you will not need to modify any operator definition file.

4.2.2 Icon Graphics

Under the `libpath` directory (defined in the configuration file), usually

```
$WITHOME/lib/icons
```

there is a directory called `icons`, which in turn contains a sub-directory for each operator library. For example, there are directories `io`, `point`, etc., for the built-in WiT libraries. Each of these directories contain a separate icon file for each of the operators in that library.

4.2.3 Modifying Icon Graphics

Currently WiT only accepts icon files in XBM format. If you do not have a program that can produce this format, create an icon in any of the formats that the **rdImage** operator in WiT understands. Run WiT and use **rdImage** to read in the icon, then use the **wrImage** operator to write it out in XBM format. Make sure the name of the icon is correct: it should be the same as the operator name, with no file extensions.

Simple changes can be conveniently done within WiT with the WiT icon editor (§10.15). However, you may want to make major changes with a more sophisticated icon editor that can produce X11 bitmap format. For example, the OpenWindows 3 utility `iconedit` or the X utility program `bitmap` are both good programs. The size of an icon is usually 60×40, although WiT can handle any reasonable size. Notice the port positions are not defined with the icon file, but in the operator definition files in `$WITHOME/lib/*.def`. It is easiest to position ports interactively with the WiT built-in icon editor.

We suggest that you make a backup of the original icon, in case you or somebody else want to revert to the original icon. See the *Programmer's Manual* for information about making an icon for a new operator (as opposed to changing an icon for an existing operator).

4.2.4 Icon Cache

Because of the large number of operators that WiT supports, reading in all the icons that are in text can take considerable time. For this reason, WiT maintains an icon cache, normally stored as file

```
~/witicons
```

The location of this file can be changed (§13.4). New icons are parsed and stored in the cache every time WiT is run. However, if you modified an existing icon, WiT will not know about it and will still use the old icon from the cache. To force WiT to use the new icon, delete the `witicons` file. If you use the WiT icon editor, the icon cache is removed automatically. This will cause some delay the next time WiT is started, because it has to parse all the text icon files, but that will only need to be done once. The next time WiT is started, it will revert to the (now correct) icon cache.

4.2.5 Default Icon

When a new operator is created but no icon has been designed for it, WiT uses a default 60×40 icon, stored as file `$WITHOME/config/default.xbm`

You can change this icon to change the way operators without icons are displayed. If you do, be sure to remove the icon cache (§4.2.4).

4.3 Operator Properties

Each operator may have a number of parameters which control its execution, e.g. the width of a filter, name of file to read, etc. Operator parameters are simple object types such as integers, characters or strings. They cannot be complex data structures. A parameter must have a name, e.g. **filename**, **width**.

On invocation of an operator, a properties panel is presented for you to specify its properties. On the properties panel, each parameter is listed with its name followed by a data entry area such as a slider or choice list for you to enter a value.

In addition to parameters specific to the operator, the properties panel present standard properties which applies to all operators. These properties are for things such as execution control and visual appearance of the operator, they are different in demand-driven (§10.12) and data-driven mode (§10.12.1) (the operator specific parameters are identical in either mode).

4.4 Getting Help for Operators

There are many ways you can get help for an operator:

- Point at the operator icon in the operator library panel and hit the **Help** (**F1**) key.
- Point at the operator icon in the operator library panel and choose the **Help** menu item.
- Point at an instance of the operator in an Igraph and hit the **Help** key.
- Select an operator from the Operator Selector and hit the **Help** button on the Operator Selector.

4.4.1 Getting Help for an Operator Port

Sometimes an operator has so many ports that it is not obvious what each port does. Usually the graphics in the icon should indicate the functions of the ports, but when there are many ports, attempting to show port functions graphically may result in extremely complex looking icons. That would defeat the purpose of icons, which should be something simple and easy to recognize. For this reason, a help window can be invoked for an individual port by pointing the mouse at the port and hitting the **Help** key. The help window will tell you the port number and name, and the type of object that the port is expected to receive or produce.

Chapter 5

Data Objects

Objects can be images, simple scalar values such as integers or strings, or complex data structures which consist of many different fields. WiT can read, write, display, print, and send data objects across networks, among possibly different computer types. Objects other than images will be referred to as *general objects*.

Objects traveling in an Igraph are called *tokens*. A token consist of a reference to an object plus information about where it is located, how many operators require that object, etc.

5.1 Viewing Objects

You can view objects (images or general objects) produced as a result of executing an operator, or view object *files* (saved as WiT objects) directly.

5.1.1 Viewing Objects in Data-Driven Mode

There are two ways to display an object token: you can send the token to the **display** operator, or you can put a probe (magnifying glass) on a link (§6.4.6). **Display** pops up a window whenever it receives a token. If the object is an image, it will be displayed in its original size if it is smaller than the size of the display screen. If it is bigger, it is resized to fit the screen. If the object is a general object, it is displayed as formatted text.

If a probe is put on a link, then whatever travels on that link when the Igraph is running will be displayed in a pop-up window, exactly like the **display** operator.

5.1.2 Viewing Objects in Demand-Driven Mode

Objects produced by operators are automatically shown in miniaturized form on the card stack. If you want to see them at their original scale, click (press and release) the left mouse button at the card you wish to see, and the image will come up as a pop-up window.

You can also execute the **display** operator to display an image at its original size. Usually it is less convenient than simply clicking at the card. However, using **display** does offer one advantage: if you want to display more than one object, but they all have the same name on the cards (such as when they are produced by the same operator, e.g. **rdObj**), then using **display** will allow you to specify a different name for the pop-up object window. Changing the name is necessary because when WiT creates new object windows, it searches for any existing object windows with the same name. If a window is found, the contents of the window is overwritten with the new data, instead of a new window being created.

5.1.3 Viewing Object Files

Select **File/View objects...** from the menu bar. This brings up a **File** panel (Section 10.7). WiT objects have a '`.wit`' extension. Select a file and click the **OK** button. The object appears in a detached window.

5.2 Studying Images

WiT provides many utilities for examining image data:

- Report image type: unsigned 8-bit, floating-point, color, etc.
- Report image width and height.
- Report mouse position in image coordinates.
- Report pixel value at mouse position.
- Draw X and Y profiles at mouse position.
- Change image color map, including user-defined color maps.
- Adjust image contrast.

These functions are accessed from the image properties panel (§10.18). To bring up the properties panel for an image:

1. Move the mouse cursor inside an image which has been displayed in its own window (i.e. not in miniature form on a card).
2. Press and hold the right mouse button. A menu pops up.
3. Select the **Properties...** menu item. The image properties panel pops up.

5.2.1 Studying Pixel Values

To study the exact value of a pixel in an image:

1. Bring up the image properties panel (§10.18).
2. Move the mouse inside the image. The value of the pixel pointed to by the mouse is displayed in the **Value** field on the properties panel. The value depends on the image type. For example, if the image is floating-point, then the value is a floating-point number. If the image is color, then the value is an RGB triplet.

5.2.2 Plotting Pixel Profiles

To plot the pixel value variation along the X or Y axis on the image:

1. Bring up the image properties panel (§10.18).
2. Check the box **X** or **Y Show profile** check box (or both if you want). A pop-up window comes up (Figure 5.1) which plots pixel values along the X or Y axis.
3. Move the mouse inside the image. Drag with the left mouse down. The profile plot is updated as you drag your mouse. Notice also the line on the image. This is the line of pixels which is being plotted.

To remove a profile window, do one of the following:

- Uncheck the appropriate **Show profile** check box.
- Close the profile window.
- Close the image window: profile windows are destroyed automatically when the parent image window is destroyed.

5.2.3 Adjusting Image Colors

Sometimes you may want to adjust the contrast on a grayscale image to study how the pixel values are varying. Or you wish to install a pseudo-color colormap to further bring out pixel variations. See Section 5.6 for details.

5.3 Resizing Images

When images are first displayed, they are displayed in their original size. You can change the size of the image by changing the size of the window. WiT handles all window size changes in a consistent way (§8.2). You can also scale an image precisely using its properties panel (§10.18).

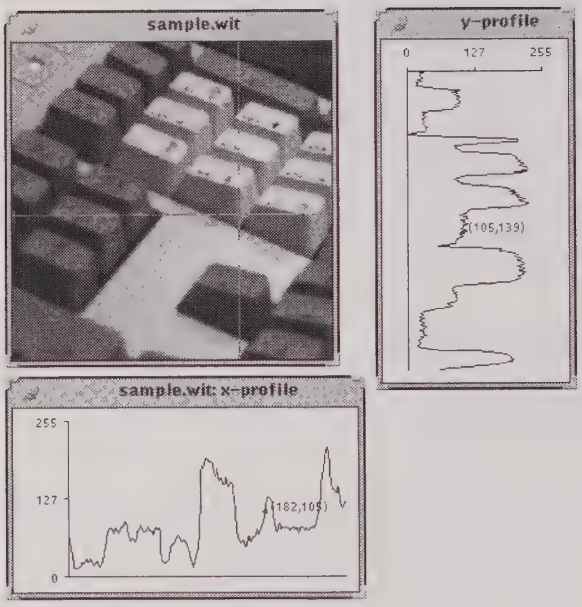

Figure 5.1: Pixel Profile Plot

5.4 Zooming and Panning Images

Sometimes you want to magnify an image without increasing the window size (i.e., make the window size smaller than the image size), so that the window displays only a portion of the image. You may then want to pan the window to look at different portions of the image. Zoom and pan are handled in a consistent manner across all windows in WiT (§8.1).

5.5 Image Types

WiT supports many different image types. Samples of all supported image types are provided in the `$WITHOME/images` directory:

Image Type	Example Image
8-bit unsigned	sample
8-bit signed	mri
16-bit unsigned	fighter
16-bit signed	brain
32-bit grayscale floating-point	moon
24-bit color	frog

With the exception of color images, all built-in WiT operators can deal with all image types. Only a subset of WiT operators can deal with color images directly.

Typically, color images are processed as follows:

1. Split the image into its RGB or HSV components with the **rgbSplit** or **hsvSplit** operator. Each of these components is an 8-bit unsigned image.

2. Process each of the RGB or HSV components with the same operators.

3. Merge the three components back using **rgbMerge** or **hsvMerge** to form a color image.

Figure 5.2 (demo Igraph `color`) shows an example of color image processing using **rgbSplit** and **rgbMerge**.

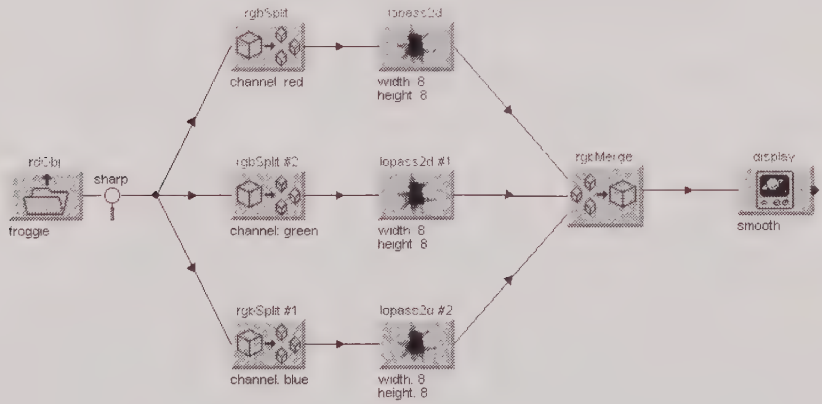

Figure 5.2: An Example of Color Image Processing

5.6 Contrast and Color Control

5.6.1 Grayscale Images

You can change the contrast on images to enhance certain features for viewing purposes. The grayscale ramp on the right of the properties (§10.18.1) is for this purpose.

You can also display a grayscale image with different pseudo-color color maps (§10.18.2).

5.6.2 Color Images

On displays with 256 colors, color images are normally displayed with a shared color map, which uses a stipple pattern to approximate colors. This allows multiple color images to be displayed correctly simultaneously. However, resolution is reduced because of the stipple effect. Alternatively, you can choose to use a private color map (using the image properties panel), which means the color map is set up so that only colors relevant to this particular image are stored. The effect is that the image is no longer stippled, so resolution is increased. But the side effect is that if there are other color images being displayed, their colors will become incorrect. You can load the correct colors for a particular image by clicking the left mouse button at the image. Switching between private and shared color maps on color images does not affect the display of grayscale images.

On displays with true color, color images are displayed as accurate as the display allows.

5.7 Printing Images

There are two ways you can print images from WiT:

1. Display the image and select the **File/Print** menu item (§8.5).
2. Execute the **print** operator with the image as input.

5.8 Studying General Objects

5.8.1 Scroll Bars

Object windows have a scroll bar on the side if the object cannot all fit within the window. You can use the scroll bar to look at different portions of the object. When the object is printed, the same portion which is visible on the screen is printed.

5.8.2 Changing Font

To change the font used on an object window, use the pop-up properties menu and select the **Font...** item to bring up a font panel. Select a font and click **OK**. the font changes applied to an object window is preserved when the object is printed, provided that the font family is supported by the printer (§8.5).

5.8.3 Saving a Displayed Data Object

You can save a displayed data object as either a text file or as a WiT object. Select the **Save as text...** or **Save as object...** from the pop-up menu.

5.9 Printing a General Data Object

There are two ways you can print general data objects from WiT:

1. Display the object and select the **File/Print** menu item (§8.5). The object is printed as it appears on the screen.
2. Execute the **print** operator with the object as input. This method has the additional option of printing the object out as text, so that the entire object can be printed and the output can span multiple pages. See the **print** operator on-line help for details.

5.10 Reading/Writing Objects and Search Paths

When WiT reads an object with operators such as **rdObj** or **rdImage**, it uses a search path to look for it. Search paths allow frequently used objects to be found regardless of where your working directory is, and without you having to specify the complete path. Search paths are set up in the configuration file (§13.3).

Writing objects is handled differently. Objects and print files are always written to the `object` directory, which can be changed with the **File/Object dir...** menu item.

5.11 Object Window Placement

In a WiT session, it is possible that many object windows are displayed simultaneously. There are three methods that WiT can use to position new object windows on the screen, controlled with the **Object placement** widget of the Setup panel (§10.4.1):

- Tile windows.
- Place windows at where the mouse is.
- Place windows randomly on the screen.

Chapter 6

Data-Driven Mode

The Igraph concept is one of the most noticeable features in WiT. An Igraph is a graphical representation of an imaging algorithm. On an Igraph, each icon represents an operation, such as reading an image file, performing a convolution, or generating a lookup table. Each operation takes some number of input objects and produces some output objects, which flow along paths specified by directed links.

Operators are represented by *graphical* icons, not just a block diagram with text inside. This makes your imaging algorithm pleasant to work with and easy to understand. The properties of each operator can be interactively modified and displayed below the icon so that the Igraph is a *complete* representation of your algorithm. Flow control operators such as *if-then-else* and *for-loop* allow you to incorporate conditional logic within an Igraph.

When an Igraph is executed, WiT dynamically shows you data objects traveling along links or being processed by operators, and which inputs and outputs are being consumed and generated. Controls for pausing, continuing, or stopping Igraph execution are available.

Igraphs present a quick way of prototyping and testing image analysis procedures without committing hours of tedious programming. With meaningful graphical icons for operators and parameter values displayed directly below each operator, an Igraph is not only the means of executing an algorithm, it also serves as concise and accurate documentation for it.

WiT also provides a different execution mode: demand-driven mode, used for quick testing of ideas. Details can be found in Chapter 7.

6.1 Running WiT in Data-Driven Mode

To run WiT in data-driven mode, do one of the following:

- Use the command line option **-mode data** (§13.1.4).
- Click the tool bar button.
- Bring up the Setup panel (§10.4.1). Choose **Data-Driven** for **Mode**.

6.2 Igraphs

In an Igraph, operators process data objects which are transmitted along links. An operator may modify an incoming object and then send it out, or simply consume it. Some operators produce objects without requiring input objects: from reading files, fetching data from a frame grabber, or creating data using a formular, etc. The behavior of an operator can be controlled by modifying its parameters (§4.3).

Links (§6.4) can branch off into an unlimited number of directions. When an object reaches a branch junction, it multiplies itself and the individual branches can be executed *in parallel*, provided that you have more than one processor available.

There is no limit to the number of operators or links in an Igraph. An Igraph can be encapsulated into a hierarchical operator (§6.7) and incorporated into other Igraphs. There is no limit to the number of nesting levels of hierarchical operators.

6.2.1 Creating a New Igraph

To create a new Igraph, do either one of the following:

- Click the tool bar button.
- Select menu item **File/New**.

The WiT workspace is cleared and any pop-up frames generated by the previous Igraph execution are taken down. Pop-up frames can be optionally retained (§10.4.2).

6.2.2 Reading an Igraph

To read a previously created Igraph, do one of the following:

- Click the tool bar button.
- Select menu item **File/Open**.

A filer window comes up for you to select an Igraph file. Igraphs have a `.igr` extension. The initial directory can be specified in the configuration file (§13.3). Afterwards, the directory is the one where the last Igraph is loaded from or saved to.

6.2.3 Saving an Igraph

To save an Igraph, select menu item **File/Save**.

To save an Igraph to a different file name, do one of the following:

- Click the tool bar button.
- Select menu item **File/Save as....**

When you save an existing Igraph to itself, WiT will do it quietly. If the Igraph is new, selecting **File/Save** has the same effect as selecting **File/Save as....**

When saving an Igraph to a different name, a filer panel comes up for you to select a file name. WiT adds the 'igr' extension automatically to the name you enter, so you do not have to type that yourself.

If you try to save an Igraph as another file name and that file already exists, WiT will ask for your confirmation to overwrite the previous file.

An Igraph is stored with most of the editing options that are used at the time the Igraph is saved. These are:

- Workspace window size.
- Grid size.
- Grid visibility.
- Pick tolerance.

With this information, WiT can restore the (possibly different) working environments that you used to created different Igraphs. For example, you probably want to use a dense but invisible grid to design a very complex Igraph with many small objects, but a large and visible grid for simple Igraphs.

6.2.4 Running an Igraph

To run an Igraph, do one of the following:

- Click any one of the buttons. These buttons all start the Igraph running, but with different speeds (§8.4).
- Select **Run/Start**.

Before an Igraph starts, all pop-up windows from the previous execution are cleared automatically. Pop-up frames can be optionally retained (§10.4.2).

The execution is performed differently depending on the run speed:

- Walk: The status indicator cycles between and . Operators turn green when they execute, and their names are reported on the status window. A one second pause is added after each operator before proceeding to the next. Token travel on links is animated slowly so that it is easy to follow the progress of the Igraph.

- Run: The status indicator cycles between and . Operators turn green when they execute, but the pause after each operator is shorter, and token travel is animated faster than **walk**.
- Sprint: The status indicator cycles between and . Operators turn green when they execute, but there is no pause added after execution. Token travel animation is turned off.
- Warp: All feedback is disabled. Even the status indicator becomes a static , thus allowing the Igraph to run absolutely unimpeded.

You cannot edit an Igraph during execution, In addition to turning green to indicate execution, other colors are used to indicate different operator states (§6.3.4).

6.2.5 Pausing Igraph Execution

To pause an executing Igraph, do one of the following:

- Click the button.
- Select the **Run/Pause** menu item.

WiT may take a while to respond to your pause request, because it has to wait till all currently active operators have finished before it can pause, and because WiT is busy executing, some window events may be lost. So you may have to try several times. If the pause request is received by WiT, the status indicator changes to . When all the active operators are finished, the status indicator changes to to indicate that the Igraph is now paused.

When an Igraph is paused, you still are not allowed to edit it. But you can study at leisure images and other objects which have popped up.

6.2.6 Resuming a Paused Igraph

To resume an executing Igraph, do one of the following:

- Click the button.
- Select the **Run/Continue** menu item.

The Igraph resumes where it was last paused. The button and the **Run/Continue** menu item are normally disabled except after an Igraph has been paused.

6.2.7 Stopping an Igraph

To stop an executing or paused Igraph, do one of the following:

- Click the button.
- Select the **Run/Stop** menu item.

WiT may take a while to respond to your pause request, because it has to wait till all currently active operators have finished before it can pause, and because WiT is busy executing, some windows event may be lost. When the stop request is received by WiT, the status indicator will changes to . When all the active operators are finished, the status indicator then changes to to indicate that the Igraph is stopped and is ready for editing. You can stop a running or paused Igraph. Igraph execution cannot be resumed once it is stopped.

6.2.8 Zooming and Panning an Igraph

The zoom and pan techniques described in Section 8.1 apply to Igraphs. In addition, you can select the **Zoom/pan...** item from the workspace pop-up menu to get a panel for incremental zooming and panning (§10.11).

6.2.9 Customizing the Grid

The grid that is shown on the workspace is not only used for decoration only. When snap is enabled (from the **Data Driven Options** panel), objects will snap to a grid point when placed or moved on the workspace. When making a link, however, operator ports and link corners have priority over the grid. For example, if you branch off from a link in the middle of one of its link segments, then WiT will choose to break the segment at a grid point. But if you branch off from near a link corner, WiT will break the segment right at the corner. This usually results in a better looking Igraph.

If you want to position objects at a different resolution, you can change the grid size (§10.4.2). If you set the grid size to 0, the grid will not be displayed at all.

Sometimes you may find the grid distracting, so you may want to hide the grid (§10.4.2). The effects of snapping are still enabled even when the grid is hidden.

6.2.10 Getting Help for Igraphs

You can get help for what a particular Igraph is supposed to do by hitting the **Help** key at an empty spot on an Igraph. A window with information about the Igraph pops up.

6.3 Igraph Operators

Usually, an operator is shown on an Igraph as an icon with the operator name at the top, parameters at the bottom, input ports on the left, and output ports on the right. Unconnected ports are denoted with solid black arrows. When connected, a port has a link attached to the location where the arrow used to be. Figure 6.1 shows a typical operator with some ports connected and some unconnected.

Figure 6.1: Typical Operator Appearance

An Igraph operator is an Igraph object (§6.6), which in turn is a graphical object (§8.3).

6.3.1 Moving Operators on an Igraph

An operator icon is a graphical object. The techniques discussed in Section 8.3.2 apply. In addition, there are techniques specific to operators which do not apply to graphical objects in general. These are explained below.

You can drag individual operator icons without first selecting them. When *one* operator is moved, all the links connected to it stretch with the movement of the operator. This allows you to shift an operator small amounts without having to reconnect all its ports. This behavior does not apply when multiple Igraph objects are moved: connected links not selected with the group will become disconnected.

Whether moving single or multiple icons, link connections are automatically made if an operator port happens to fall on a disconnected link end (shown as a frayed end ✳).

6.3.2 Cutting, Copying, and Pasting Operators

An operator icon is an Igraph object. The techniques discussed in Section 6.6 apply.

6.3.3 Changing Operator Properties

Click the right mouse button at the icon. The properties panel (§10.12.1) for that operator is displayed. Set properties as desired, then hit the **OK** button. If you hit the **Cancel** button, all changes you made on the properties are discarded.

6.3.4 Operator Color Indicators

On a color display, colors are used to indicate various operator states:

Color	Meaning
Red	selected
Green	busy
Blue	output blocked
Brown	waiting for user input

All colors are user configurable (§13.2). On displays which do not have enough colors (or monochrome), WiT will try to indicate different states as well as it can. For example, on a monochrome display, WiT uses different stipple patterns to indicate selected and busy operators.

6.3.5 Parameters as Inputs

Sometimes it is necessary to make an operator parameter an input. For example, you may want to set the filter width of a low pass filter dynamically depending on the contents of the image itself. So you would like the filter width to be an input rather than a fixed value. Such an input can then be connected to a link to get its data. WiT produces an input port for a parameter if the parameter field in the property panel of the operator is empty, or if the **Input** choice (§4.3) is selected for the parameter. Parameter input ports are created on the *bottom* of the operator icon.

All hierarchical operators are of type **fire-on-any** (§12.4.2). See Section 12.4.3 for special behavior of parameter input ports on such operators.

6.3.6 Getting Help for Operators in an Igraph

You can get on-line help about any operator in an Igraph by hitting the **Help** key when the mouse is pointing at it. See Section 4.4 for more ways to get help for an operator.

6.4 Igraph Links

Igraph links represent the paths data takes when an Igraph is executed. Links are always uni-directional, the direction of data-flow indicated by an arrow on the link. A link is an Igraph object (§6.6), which in turn is a graphical object (§8.3).

6.4.1 Creating Links

To create a straight link between two operator ports:

1. Point the mouse at the first operator port. While the mouse is there, do either one of the following to connect a new link to the port:
   - *Double click* the left mouse button.
   - Hit the period ('.') key on the keyboard.

2. Move your mouse to point at the other operator port. While the mouse is there, do either one of the following:

   - *Double click* the left mouse button.
   - Hit the period ('.') key on the keyboard.

To create a link with corners between two operator ports:

1. Start the link as before.
2. Move mouse to position of first desired corner, then *single* click the left mouse button. A corner is created. The link now rubber bands between the mouse and the corner.
3. Repeat for additional corners.
4. If a corner is mistakenly placed, click the *middle* mouse button to backtrack one corner. If you are already at the beginning of the link, the link creating session will be terminated.
5. Terminate the link by attaching it to another operator port as before.

You can abort the link creation session at any time by hitting the ESC key on the keyboard.

Once the link is attached at both ends, WiT will attempt to make the link look pretty by snapping all link segments to align with a port, junction, or the grid. This can be overridden later (§6.4.2).

You can only connect an input port to an output, or vice versa, but not two ports of the same type.

You can connect a link to the middle of another link (§6.5), or to a frayed link end ✳ (§6.4.5).

Since double clicking deselects all previously selected objects, the period ('.') key method of starting a link is particularly useful when you want to create a link without losing your current selection (very useful in tracing (§10.1.3), for example).

6.4.2 Editing Links

Once a link has been created, it can be adjusted and moved. A link is an Igraph object. The techniques discussed in Section 6.6 apply.

When you adjust links, WiT no longer tries to make the wire pretty by forcing it to lie along grid lines or multiples of 90 degrees.

6.4.3 Changing Link Properties

You can change properties of a link by clicking the right mouse button at it, which brings up the properties panel (§10.13). Unlike operators, all links have a common set of properties, but the property values may be different.

6.4.4 Link Direction

The direction of a link is often implicitly determined by the kind of ports it is connected to: from an output to an input port. If you attempt to connect a link between two input ports, or between two output ports, WiT will warn you and refuse to make the link. When the direction of a newly created link cannot be determined (e.g. a branch between two links), the data is assumed to flow in the direction that the link is created, i.e. following the order in which the points on the link are entered.

6.4.5 Frayed Link Ends

It is possible for a link to become disconnected from everything (for example, when you delete a node which has some connected links). Such dangling link ends are displayed with a frayed end ✳ on the Igraph. You can connect a link to a frayed end by double clicking the left mouse button at the frayed end to start wiring, or you can connect an operator port to it by simply dropping an operator icon on top of the link end, provided the direction of the operator port is compatible with the direction of the link.

6.4.6 Probes

A probes on a link is a convenient mechanism for you to inspect data when an Igraph is running. A probe can be added or deleted from the link properties panel (§10.13). A probe behaves in exactly the same way as the **display** operator, but is more convenient and compact. When data flows along a link which has a probe, a pop-up window is created to display the data. The title of the window is the name of the link.

6.4.7 Getting Help for Links

Hit **Help** on a link, you will get general information about links, such as how to create a link between two ports, split a link to form a junction, edit link corners, etc.

6.5 Link Junctions

A link can have any number of branches coming from it, or any number of branches going to it, or both. The junction where links join or split from is shown with solid black dot on an Igraph. Figure 6.2 shows some examples of link junctions. Outgoing links from a junction are used to make multiple instances of a token. Incoming links to a junction can be used for synchronization (§12.4.4), or to simply feed different tokens to the same destination.

When a token arrives at a a junction which has multiple outgoing branches, the junction will hold the token if any of the ports that the branches are leading to is not ready to receive a token. During this time the junction will not accept any new tokens. Once all ports have received the token stored in the junction, the junction removes that token from its storage and resumes to accept new tokens.

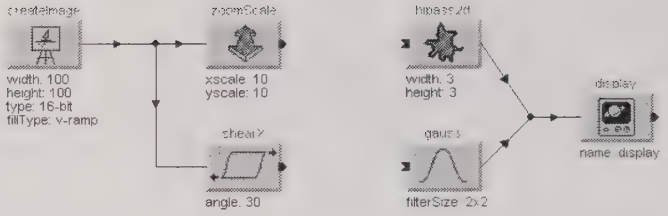

Figure 6.2: Example Use of Link Junctions

6.5.1 Creating a Junction

To create a junction and a link branch, point at the location within a link you want the branch to come off, and double click the left mouse button (or use '.'). Make sure no operator ports are within the pick region, otherwise WiT will think you want to connect to the port.

If the split is successful, a rubber band wire is created between the mouse pointer and a newly created junction on the link, shown as a large dot.

You can have as many branches from a link as you want. This is how multiple instances of an object can be sent to different operators. Again, you can use the middle mouse button to back up from any incorrect corner points that you have created. When you back up all the way to the link junction, the junction is deleted.

Link Snap Priority when Creating Junctions

When a link is split, WiT uses the following rules, in order of preference, to put the junction at a visually pleasing position:

1. Existing link junctions.
2. Existing link corners.
3. Grid intersection point.

6.5.2 Deleting a Junction

Junctions are not deleted directly. They are deleted when all branches from a link have been deleted. When a link is deleted from a junction, WiT employs some intelligence to determine whether the junction can be deleted or not. If a junction has only two links connected to it after a link is deleted, and one link is pointing towards the junction and the other one is pointing away from it, then the junction can be deleted, otherwise it is retained. It is therefore possible for a junction to have only

links bringing data to it but no links carrying them away, or *vice versa*. Such a construct of course is not useful, but is a valid state during the Igraph design process.

6.6 Manipulating Igraph Objects

There are two kinds of Igraph objects: operator icons and links. They can be selected, moved, copied, and pasted.

6.6.1 Selecting Igraph Objects

An Igraph object is a graphical object. The techniques discussed in Section 8.3.1 apply. An operator icon turns red when it is selected. You can select multiple icons and links. You can select all icons and links in an Igraph with the menu item **Edit/Select all**.

6.6.2 Cutting Igraph Objects

If you want to cut one or more Igraph objects, you need to select them first. Then do one of the following:

- Choose menu item **Edit/Cut** from the menu bar.
- Choose menu item **Cut** from the workspace menu.
- Click tool bar button.

Cut objects are saved in a buffer. They can be pasted later, possibly to a different Igraph. Objects which have been previously added to a sub-graph can be cut, but objects which are originally in a sub-graph cannot be deleted.

Links can be pasted to a sub-graph. Operators cannot be pasted to a sub-graph.

6.6.3 Copying Objects

To copy selected objects to the buffer, do one of the following:

- Choose menu item **Edit/Copy** from the menu bar.
- Choose menu item **Copy** from the workspace menu.
- Click tool bar button.

Copied objects can be pasted later, possibly to a different Igraph, or from a sub-graph to the top-level Igraph. Links can be pasted to a sub-graph. Operators cannot be pasted to a sub-graph. Any object in a sub-graph can be copied.

6.6.4 Pasting Objects

To paste objects in the buffer to an Igraph, do one of the following:

- Choose menu item **Edit/Paste** from the menu bar.
- Choose menu item **Paste** from the workspace menu.
- Click tool bar button.

When you paste, WiT automatically goes into drag mode so that the objects can be placed interactively.

Objects can be pasted from one Igraph to another, or from a sub-graph to the top-level Igraph. Links can be pasted to a sub-graph. Operators cannot be pasted to a sub-graph.

6.6.5 Duplicating Objects

Duplicating is the same as copying followed by pasting. It simply saves you some steps. There are two ways to duplicate currently selected Igraph objects:

- Choose menu item **Edit/Duplicate** from the menu bar.
- Choose menu item **Duplicate** from the workspace menu.

6.6.6 Pick Tolerance

When you need to select something on an Igraph, the size of the square cursor tells you how accurate you need to be. You can change the pick tolerance from the **Data Driven Options** panel (§10.4.2). When you change the pick tolerance, the size of the square cursor will change accordingly.

When objects are close together, you might want to lower the pick tolerance to allow finer, pin-point control. Conversely, you may set a coarse pick tolerance on sparse Igraphs so that it is easier to pick things. The pick tolerance is not related to snap or the grid size.

6.6.7 Cut and Paste Across Igraphs

Sometimes you may want to incorporate in part or in full one Igraph into another. This is easily done because the copy buffer (filled when you copy something using the **Copy** item from the workspace pop-up menu) is preserved when you load a new Igraph. So all you need to do is load the Igraph that you want to copy from, select the operators and links you want, and copy them (§6.6.3). Then load the Igraph you want to copy the objects to (or clear the workspace for a new Igraph), and paste the objects (§6.6.4).

6.6.8 Object Names

Every node and link on an Igraph must have a name. By default, the first instance of an operator is given the operator name . Any more occurrences of the same operator will have a number appended to the base operator name. You can rename an operator to whatever name you want, provided that same name is not already in use. If it is, then a number will be appended to the name to make it unique. If you delete the name altogether, WiT automatically replaces it with the default base name: a single underscore character ('_').

Most printable characters can be used in instance names, including space (' '), but you must avoid the backslash ('\', used internally for hierarchical information) and the number ('#', used for instance numbers) characters. Also, WiT does not display any name that starts with the underscore character ('_'). This is the mechanism used by WiT to selectively hide names of Igraph objects. So do not use it as the first character of a name if you want that name displayed.

Link names are assigned following the same rules. The only exception is that by default, link names are not displayed (they start with an underscore).

By default, instance numbers are shown. However, when dealing with large Igraphs, instance numbers can make the Igraph look cluttered and are best turned off. In addition to specifying individual instance names, you can control how instance names are generally displayed using the **Show names** control item on the Data-driven Options panel. They can be turned off completely (**None**), with only the base name showing (**Base**), or with the instance number showing as well (**Instance**).

6.7 Hierarchical Operators and Sub-Graphs

WiT allows you to make an Igraph into a new operator, which can then be incorporated in another Igraph. Such operators are called hierarchical operators (as opposed to *primitive* operators written in C). You can use hierarchical operators to group a series of operations that you often use, or to make a large Igraph easier to handle and understand. A hierarchical operator is similar to a subroutine in a programming language. It allows a useful algorithm to be incorporated in other algorithms. If a user is not interested in the internal structure of a hierarchical operator, he can simply use it like he would any other operator, in much the same way that a subroutine in a programming language can be used as an extension to the language.

A sub-graph can in turn contain hierarchical operators. There is no limit to the number of levels on an Igraph. There is also no limit to the number of input or output ports on a sub-graph, or the number of promoted parameters.

A hierarchical operator is shown with a red dot on its upper right corner. The hierarchical operator is called the *super-node* of its associated Igraph, while the Igraph is called the *sub-graph* of the hierarchical operator.

6.7.1 Making a New Hierarchical Operator

To create a hierarchical operator, select the **Graph/Make operator** menu item. A panel like Figure 6.3 comes up.

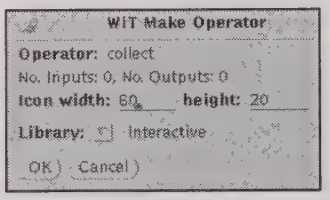

Figure 6.3: Make Operator Panel

The size of the operator icon is automatically set so that all input and output ports can comfortably fit on the left and right sides of the icon respectively. The locations of these ports are also computed automatically, and follow the order in which they appear in the Igraph. You can alter the default icon size if you want, but normally you should not reduce the height of the icon, since that will leave insufficient room for the placement of ports.

Select the operator library that you want the new operator to be put in, then click **OK**. A new operator, with the same name as the Igraph that is currently being edited, will then appear in the library that you specified.

If you want to delete a hierarchical operator from a library, move the mouse inside the library panel and to the operator you want to delete, then bring up the pop-up menu (right mouse button). Select the **delete** item. You will be given a warning before WiT actually removes the operator. The corresponding Igraph will *not* be removed, though.

Sub-graph input and output ports are operators found in the **Dataflow** library. You should try to give ports meaningful names, since they will be used to label the super-node ports, and will come up on the help panel when the user asks for help.

The location of the ports on the super-node is determined by their relative vertical positions in the sub-graph. Input ports will be placed on the left, output ports on the right. If two ports of the same type are at the same vertical position, then the one on the left (smaller horizontal coordinate) will be assigned first.

WiT tries to set the port positions as wide apart and evenly distributed on each icon side. In most cases, you should use the icon editor (§10.15) to check the port positions, and change them if necessary.

When you add or delete ports from a hierarchical Igraph, WiT maintains the positions of all previously defined port positions. Only new ports are given automatically computed positions.

6.7.2 Sub-Graph Ports

Each port (both input and output) on a hierarchical operator must have a corresponding port *operator* in its sub-graph. Port operators look like ordinary operator ports, but are hollow and a bit larger. Data association between a hierarchical operator and its sub-graph is established through port *names*. The position of the port operators is not important as far as data-flow is concerned.

When a hierarchical operator is first created, the default icon size is just big enough to show all input port on the left of the icon, and all output ports on the right. The position of the super-node ports will be relatively the same as the port operators on the sub-graph. Again, this is just for convenience. The super-node ports and port operators can later be moved (using the icon editor (§10.15)) without affecting the correct execution of the sub-graph.

6.7.3 Promoting Parameters

You can promote a parameter of any node or link in a sub-graph to the super-node, i.e., they are linked to the parameters of the super-node. There is no limit to the total number or parameters promoted.

To promote a parameter, click the **Promote** button on the properties panel (operator or link). A panel comes up for you to enter the names of the promoted parameters to be used on the super-node. If you leave the name field blank for a parameter, then that parameter is not promoted. If that parameter was previously promoted, it will be un-promoted. Promoted parameters are shown in reverse print on an Igraph.

6.7.4 Changing Operator Parameters in a Sub-Graph

Even if the parameters in a sub-graph are not promoted, you can still change them by bringing up the properties panel of the node in the sub-graph whose parameters you want to change, and making the changes there.

Since it is possible that hierarchical operators are incorporated in many Igraphs (just like procedures can be called from different programs), the changes that you make to either promoted or unpromoted parameters will only affect that particular invocation of that hierarchical operator. This is analogous to calling a true function (a function without side effects) in a conventional program with different parameters: it does not affect the return values of other calls to the same function.

On the other hand, if you actually load and edit the Igraph associated with the hierarchical operator itself, and change some parameters there, then every invocation of that operator will be affected, unless the parameter values have been overridden. This is analogous to changing the code or initial values of a function: it affects subsequent calls to the function.

6.7.5 Modifying a Sub-Graph

You are *not* allowed to modify the existing *structure* of a sub-graph. For example, you cannot delete a node from a sub-graph, or change the path of a link. The reason is that sub-graphs can potentially be incorporated into many Igraphs, so modifying a sub-graph may lead to unexpected behavior in other Igraphs. If you really want to modify a sub-graph, you have to exit the current top level Igraph and read the sub-graph in as the top level Igraph, and make the changes there.

Even though you cannot delete objects in a sub-graph, you are allowed to select objects in a sub-graph, and they will be highlighted just as when the objects are in the top level Igraph. This is useful for example when you have a complex link network and you want to see the path of a particular link.

Moreover, you are allowed to *add* links (but not operators) to a sub-graph. Links thus added only apply to one particular instance of the sub-graph. For example, suppose you have two Igraphs, *G1* and *G2*, both of which include a sub-graph *SG*. Suppose you load *G1* as the top level Igraph, then add some links to the *SG* sub-graph and save *G1*. The instance of *SG* which is part of *G2* will *not* have the new link.

Such sub-links can be edited like top level links. They can be moved, adjusted, and deleted.

6.7.6 Copy, Paste, and Undo Across Sub-Graphs

You can copy objects from one sub-graph and paste it to another (including the top level Igraph). If you have selected objects from different Igraphs and then execute a copy, duplicate, or cut, the action only applies to the objects within the Igraph which you invoked the action. But you can copy objects from one graph and paste it to another. You are only allowed to paste *operators* to the top level Igraph, not sub-graphs.

Chapter 7

Demand-Driven Mode

Although Igraphs are extremely powerful and easy to use, there are times when you just want to apply one or two operators to some data objects. For example, you may want to read an image and transpose it, then save it back. Having to build a tiny Igraph and then run it is inconvenient. Especially when you know you will never use the Igraph again. So WiT provides a demand-driven mode of execution. In this mode, operators that you select are executed and the results shown immediately. Demand-driven mode is useful in exploring and understanding the rich set of imaging operations WiT offers, as well as debugging any new operators that you have developed yourself.

In demand-driven mode, the WiT workspace has a number of rectangular regions which are meant to resemble the back of playing cards (Figure 7.1). The playing card analogy is appropriate because these cards often need to be shuffled around to control operator inputs. See Section 4.1 for information regarding how to execute operators in demand-driven mode.

7.1 Running WiT in Demand-Driven Mode

To run WiT in demand-driven mode, do one of the following:

- Use the command line option **-mode demand** (§13.1.4).
- Click the tool bar button.
- Bring up the Setup panel (§10.4.1). Choose **Demand-Driven** for **Mode**.

7.2 Demand-Driven Mode Cards

The number of cards in the workspace is the number of memory cells available for displaying operator outputs (data objects, of which images are a subset).

The cards behave as a last-in first-out stack: output data objects are pushed on the left of the stack as they are produced (i.e. the last output object will end up as the left-most card), and input objects

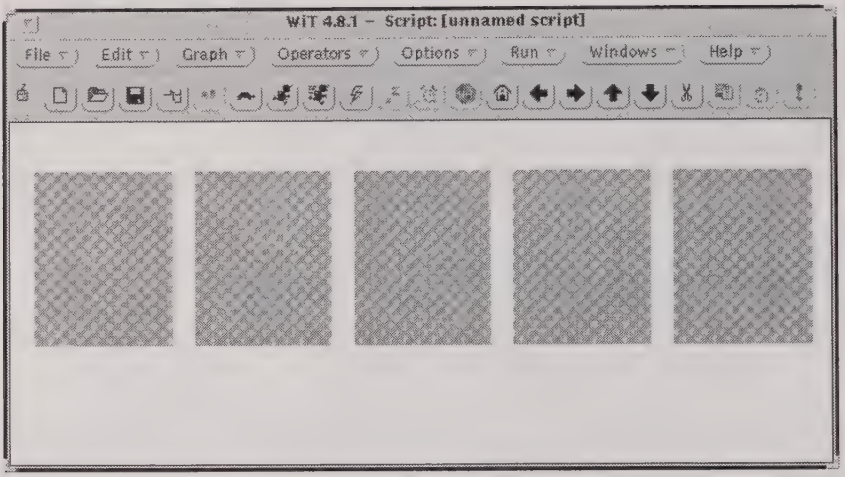

Figure 7.1: WiT in Demand-Driven Mode

are read beginning from the left of the stack (i.e. the first input object is fetched from the left-most card).

Each card has a number on the upper right corner. This number is used only as a visual cue for you to keep track of where each card is. It is the physical position of the card, not the card number, that is used to determine which cards are loaded to the inputs of an operator.

When fetching input objects, only objects of the required type will be considered. For example, if the left-most card (not necessarily number 0) is an integer object and the next card is an image object, and we wish to execute an operator which requires an image as input, then it is the second card which will be assigned to the operator input.

WiT will not use a card more than once in the execution of an operator. If you want to feed the same object to an operator more than once, make a copy of the card.

7.2.1 Displaying the Contents of a Card

For images and simple objects, such as single integers or strings, the data object is shown directly on a card. Images are scaled down to fit the card. Long strings may be truncated.

To display the data object at full size, click (press *and* release) the left mouse button at the card you want to display. The card moves momentarily down and then back up to its original position (see Section 7.2.3 for an explanation of this behavior). The object is displayed in a pop-up window of appropriate size. The label on the pop-up window is inherited from the label on the bottom of the card.

7.2.2 Card Labels

On the bottom of each card is a label. This label is important because it informs you about where the object comes from. It is also used to label pop-up object windows when you display objects at full size (§7.2.1).

The label is adopted from the operator which produced the object with the following rules:

1. The output port name, if it is defined.
2. The name of the operator. If the operator has more than one output, the port number is appended to the name.
3. If an existing card has the same name, an instance number is appended to the name.

7.2.3 Moving Cards

To control what data objects to feed the next operator, you need to move the cards around, because objects are fed to an operator starting from the left card.

To move a card:

1. Point the mouse at the card you want to move and press and *hold* the left mouse button.
2. The card moves to a lower position, and an indicator arrow appears to indicate the insertion point that the card will go to (Figure 3.14).
3. With the left mouse button held down, move the mouse until the indicator arrow is at the desired position, then release the button.
4. The card which was placed at the bottom moves to the position to the right of the indicator arrow, and all the cards to the right of the inserted card are pushed to the right.

If you release the button while the indicator arrow is pointing to the original position of the card (this is the same as clicking the left mouse button on the card), WiT will take that as a request to display the object in a detached window.

7.2.4 Deleting Cards

Only the left-most card can be deleted (except when you clear all the cards). If you want to delete any other card, move it to the left-most position first.

To delete the left-most card, do one of the following:

- Click the button.
- Select **Edit/Cut** menu item.

- Select **Cut** from the workspace menu.

To delete all the cards, do one of the following:

- Click the button. This has a side effect of clearing the execution history too, so beware.
- Select **File/New**. This also has a side effect of clearing the execution history.
- Select **Clear** from the workspace menu. This does *not* clear the execution history.

7.2.5 Copying Cards

Only the left-most card can be copied. If you want to copy any other card, move it to the left-most position first.

To make a copy of the left-most card, do one of the following:

- Click the button.
- Select **Duplicate** from the workspace menu.

The copied card is placed at the left-most position.

7.2.6 Changing the Number of Available Cards

The number of cards is usually set small because each retained object can potentially require a large amount of memory. When the card stack is full, and a new object is produced and pushed on the card stack, the oldest object is removed from the stack and any memory associated with the object is freed automatically. Depending on the amount of memory available and the particular algorithm you are trying to perform, you may want to change the number of available cards.

To change the number of available cards:

1. Select the **Options/Demand-Driven** menu item. The Demand-Driven Options panel (§10.4.3) comes up.
2. Select the desired number of cards on the panel.
3. Close the panel.

7.3 History Logging

Every operation you perform (including moving the cards around) in demand-driven mode is automatically logged. A demand-driven execution history can be saved to a file, read back, and re-executed. Demand-driven histories are saved with a '`.wdh`' (which stands for WiT Demand History) extension.

7.3.1 Reading a Demand History File

To read a previously created demand-driven history file, do one of the following:

- Click the button.
- Select the **File/Open** menu item.

A filer window comes up for you to select a history file. Demand history files have a '`.wdh`' extension. The initial directory can be specified in the configuration file. Afterwards, the directory is the one where the last history file is loaded from or saved to.

7.3.2 Saving a Demand History

To save the current set of demand-driven actions, select the **File/Save** menu item.

To save the current set of demand-driven actions to a different file, do one of the following:

- Click the button.
- Select the **File/Save as...** menu item.

When you save an existing demand history to itself, WiT will do it quietly. If the history is new, selecting **File/Save** has the same effect as selecting **File/Save as...**.

When saving a history to a different name, a filer panel comes up for you to select a file name. WiT adds the '`.wdh`' extension automatically to the name you enter, so you do not have to type that yourself.

If you try to save a history as another file name and that file already exists, WiT will ask for your confirmation to overwrite the previous file.

7.3.3 Running a Demand History

To run the currently loaded demand-driven history file, do one of the following:

- Click any one of the buttons. These buttons all start the history running, but with different speeds (§8.4).
- Select the **Run/Start** menu item.

Speed selection only affects the delay between each step when a script is replayed. Speed selection does not affect manual execution.

7.3.4 Editing a History File

There is little support for editing a history file, since demand-driven mode is intended for short sequences of operators to test out simple ideas, not long complex programs. History files are text files, so you can use a text editor to edit a history file. See Section A.1 for a description of the file format.

Chapter 8

General Techniques

Every effort has been made to make WiT a user-friendly program. Consequently, many common operations, such as zooming and panning, or choosing a font, which are needed in different situations, are done in a similar way.

Sometimes such shared techniques involve some combinations of the mouse and/or keyboard. For example, to magnify a portion of a window, whether the window is the workspace, an image, or a 3-D plot, the same combination of mouse button and motion is used.

In other cases, a special panel (window) is used. For example, when selecting a color for the background of the workspace, or a color for an image colormap, the same **Color** panel is used.

This chapter discusses those techniques that do not present itself as a panel. Those are discussed in Chapter 10.

8.1 Zooming and Panning with the Mouse

Most windows in WiT can be zoomed and panned. For example, you may find the default icon sizes too large when designing a large Igraph, so zooming out (making the icons smaller) is more convenient. Or when you are examining images, you may want to study the image on a larger scale, but you do not want to make the window bigger, which may take up too much room on your screen. Zooming changes the scale of whatever the window contains without changing the window size. WiT provides you with a consistent and convenient way to do zooming and panning on all windows where such actions are appropriate.

8.1.1 Zooming

To activate zooming, press and hold down the shift key on the keyboard, then press and hold the left mouse button at the upper left corner of the area that you wish to magnify. When you move your mouse to the *right* of the position that you initiated the zoom, the mouse cursor changes to (a magnifying glass with a '+' sign). If you move to the left, the cursor will change to (magnifying

glass with a '-' sign). The '+' sign indicates that if you release your mouse button now, objects will appear larger (zoom in), The '-' sign means the opposite (objects smaller, zoom out).

When zooming in, the area within the rubber band rectangle will be magnified to take up the entire window. When zooming out, the entire window will be shrunk to fit into the rubber band rectangle.

8.1.2 Panning

You pan the contents of a window by dragging with the middle mouse button and shift key down. Notice the cursor changes to a truck to indicate a move operation.

8.1.3 Restoring to Original Scale and Position

Windows which support zooming and panning have a **View all** item on their pop-up menu, displayed when the right mouse button is pressed within the window. Choosing this item will cause the window contents to be scaled such that all objects are visible. For example, when viewing an image, the image will be scaled so that it exactly fills the window. When applied to an Igraph, operators and links will be scaled so that all objects exactly fill the available window area.

The **Normal** item (if it exists) in a pop-up menu will restore all objects at original scale.

8.2 Resizing Windows

When a WiT window that displays any graphics (Igraph, image, etc) is resized, the contents of the window will be scaled according to the new window size. One exception is windows which display general objects (§5.8). They have scroll bars and the font size is controlled with a font panel.

For example, when you display a 64×64 image, the window size will default to 64×64 when it is first displayed, i.e., the image exactly fills the window. If you then resize the window to a size of 256×128 pixels, the displayed image will be scaled so that it still fills the new window exactly. Which means every pixel in the image will occupy a 4×2 rectangular area on the screen.

One exception to this rule is the main WiT workspace. The workspace is a design area, a resize of this window does *not* change the scale of the contents, but simply alters the size and shape of the design area.

If the contents of a window has been zoomed or panned, then resizing the window will preserve the *relative* scale *and* offset between contents and window. For example, suppose you have a 128×128 image. When it is first displayed, the window size will also be 128×128. Suppose you have zoomed in on this image so that now you are looking at an area of 64×64 of the image. Each pixel in the image therefore occupies 2×2 pixels on the screen. Further suppose that you have panned the image so that pixel (50, 50) of the image is at the upper left corner of the window. If you now resize the window to a size of 256×64, pixel (50, 50) of the image will still be at the upper left corner of the window. Also, the new window will still show the same 64×64 portion of the image, so that each pixel in the image will now occupy a 4×1 area on the screen.

8.3 Graphical Objects

WiT attempts to be as consistent as possible in its treatment of similar graphical objects. For example, operators and links in an Igraph, or a graphical object used with the **getData** operator (Section 9.1), can all be considered graphical objects, since they have shape and size, and so can be moved, adjusted, deleted, etc.

8.3.1 Selecting Graphical Objects

To select a graphical object:

1. Clicking (press and then release) the left mouse button while pointing at the graphical object. The object turns red when it is selected (stippled on a monochrome).
2. If you selected the wrong object by mistake, selecting another object will cause the previous one to be deselected.
3. If you want to *add* an object to the selected list, click the *middle* mouse button.
4. If you want to deselect an object from the selected list, click the middle mouse button on the already selected object.

To select a number of objects within an area:

1. Move the cursor to some blank spot, then press and hold the left mouse button.
2. Drag your mouse. A rubber band rectangle is drawn with one corner pinned to the position where you first clicked the mouse. The other corner is tied to the mouse position.
3. Drag the mouse until all objects you want to select are *completely* within the rubber band rectangle, then release the mouse button.

8.3.2 Moving and Adjusting Graphical Objects

Where appropriate, graphical objects (operators, links, or graphical objects), can be moved around or their sizes and shapes can be adjusted.

To move an object:

1. Select the object you want to move.
2. Press and hold the left mouse at the selected object.
3. Start dragging your mouse. The object moves along with the mouse. The cursor changes to a truck to indicate you are moving objects.

4. Drag the mouse until the object is at the place where you want it, then release the mouse button.

To move multiple objects:

1. Select all the objects you want to move.
2. Press and hold the left mouse at *any* of the selected objects.
3. Start dragging your mouse. All the selected objects move along with the mouse. The cursor changes to a truck to indicate you are moving objects.
4. Drag the mouse until the objects are at the place where you want them, then release the mouse button.

8.3.3 Adjusting Graphical Objects

Some objects, such as Igraph links and polygon graphical objects, can be adjusted (moving the link corners or polygon vertices). Others, such as Igraph operators, are fixed in both size and shape and cannot be modified.

Where applicable, to adjust an object:

1. Make sure the object you want to adjust is *not* selected.
2. Press and hold the left mouse button at the feature (link corner, polygon vertex, etc.) which you want to modify.
3. Start dragging your mouse. The cursor changes to a screw driver, and the feature rubber bands to the position of your mouse.
4. When the feature is at the position you want it to be, release the mouse button.

8.4 Selecting Run Speed

To adjust the speed (§6.2.4) at which an Igraph or demand history runs (§7.3.3), do one of the following:

- Click any one of the buttons. The speed is set automatically and then the Igraph or demand history is executed. is slowest and is fastest.
- Select **Options/Setup** to bring up the Setup panel (§10.4.1). Select **Walk**, **Run**, **Sprint**, or **Warp**. **Walk** is slowest and **Warp** is fastest.

8.5 Printing

Most graphics windows in WiT can be printed either directly to a printer or to a file. You can print a WiT image (§5.7) or general object (§5.9) directly using the **print** operator.

To print some windows, select the **File/Print...** menu item to bring up the **Print** panel (§10.20). Currently only PostScript is supported.

Most graphics windows in WiT can be selected for printing. If you bring up the pop-up menu in a graphics window, you should see that the first item is **Select**. Choosing this item will cause a check mark to be displayed on the upper left corner of the window. When **Frames to print** is set to **Selected** on the **Print** panel, only frames with the check marks will be printed.

Normaly, when you click the **OK** button, the print job will be initiated quietly. However, the first time you print something, WiT will bring up the printer setup dialog so that you can set up options such as paper orientation, which printer to use, etc.

If you wish to change printer configuration after the first print job, you can bring up the setup dialog again with the **File/Print Setup...** menu item.

8.5.1 Printer Fonts

Currently only the standard PostScript font families **Times-Roman**, **Helvetica**, and **Courier** are supported. All other fonts are mapped to one of these families, whichever one is closest in appearance. For example, **Lucida** is mapped to **Helvetica**.

Chapter 9

Interactive Operators

Some operators in the **Interactive** library allow you to interact with the operator. Several of these, such as **getData**, provide such extensive features that not all of their functionalities are described in the on-line operator help files, but are explained here instead.

9.1 GetData

The **getData** operator allows you to place graphical objects: points, rectangles, text, etc, to specific positions on an image. It accepts an image as input, and it displays the image together with a **getData** properties panel when the image is received (Figure 9.1). You can then enter graphics objects interactively on the image. Objects can be given different colors or, with text strings, fonts. They can be adjusted, moved, copied, etc., after they are entered.

When an image with graphics objects is printed (§8.5), the graphics objects are printed with the image, with the correct colors if a color printer is used.

9.1.1 Entering Graphical Objects

The **Type** button menu in the **getData** panel allows you to choose among the various graphical objects that you can enter, or to select and edit objects already entered with the **Select** item.

Because each graphic object type is so different from the others, the input procedures for each are necessarily different. Following is a description of how each graphical object type is entered. **GetData** also reminds you of what you should do next in the WiT status window.

Select	Select or edit existing objects.
Point	Each click defines a point.
Line	Press and hold left mouse button to define first end of line, then drag to other end of line and release button.
Polyline	Click (press and *release*) left mouse button to define first point. Each subsequent click defines a corner. Terminate with double-click.

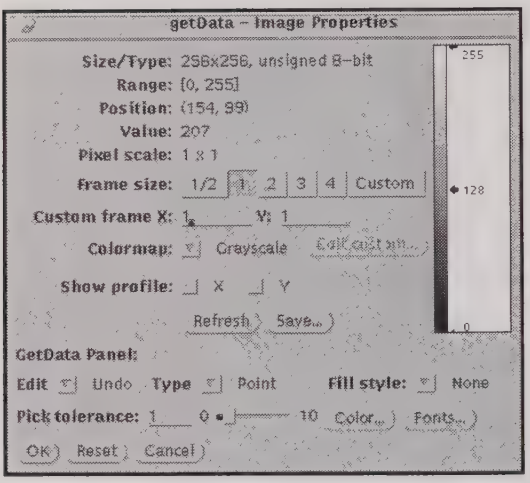

Figure 9.1: GetData Panel

Rectangle Press and hold left mouse button to define first corner, then drag to second corner and release button.

Circle Press and hold left mouse button to define center of circle, then drag to desired size and release button.

Text Click at where you want text to begin. You will see a vertical bar (the text cursor) appear. Enter text string from keyboard. Click left mouse button or hit escape key to terminate text entry.

9.1.2 Selecting, Moving, and Adjusting Objects

When you enter graphic objects, you are almost certain to make mistakes. The **getData** operator is in fact a miniature drawing program. It allows you to cut, copy, paste, move, and adjust objects which you have entered.

To edit a graphical object, you need to enter select mode first: choose the **Select** item from the **Type** menu.

WiT provides a common set of techniques (§8.3) to manipulate all graphical objects, which include Igraph operators and links, as well as all the object types supported by **getData**.

9.1.3 Edit Menu

The **Edit** menu provides these functions:

Undo Undo the last edit action.
Cut Cut the selected objects.
Paste Paste deleted or copied objects.
Copy Copy selected objects to buffer.

9.1.4 Fill Patterns

You can choose to fill graphical objects with solid color, or have them appear hollow. This is done via the **Fill** choice widget. After you change the **Fill** choice, all subsequently created closed shapes will adopt the new fill choice.

To change the fill choice of an existing object, select it first, then choose the fill choice you want.

9.1.5 Fonts

GetData allows you to associate a specific font (family, style, and size) to each text string you enter. To change the font of the text you are currently entering and all subsequent text, click the **Font...** button from the image properties panel. This brings up a **Font** panel, which is a general tool for specifying fonts (§10.9). Select a font and click the **OK** button on the **Font** panel.

To change the font of an existing text object, select it first, then choose the font you want.

There is no limit on the number of text strings with different fonts that can appear on each image.

9.1.6 Object Colors

Different colors can be assigned for each graphical object. To change the current color, click the **Color...** button. A **Color** panel pops up (§10.10). Select a color and click the **OK** button on the **Color** panel. All selected objects will adopt the new color, as well as all subsequent graphical objects you enter. There is no limit on the number of colors assigned to graphical objects that can appear on each image. Because WiT uses only a small number of colors from the window manager (§13.2), the color that you see for the graphical objects may not be *exactly* the color you specified, but the exact color *is* stored internally with the objects. If you later view the same objects on a true color display, then you will see the exact colors.

9.2 Surface

The **surface** operator draws a gray scale image as a three dimensional surface, with the intensity of the image as the height of the surface. Figure 9.2 shows an example of **surface** and its properties panel. (The properties panel is brought up from the window pop-up menu, as in image windows). Surfaces can be printed (§8.5).

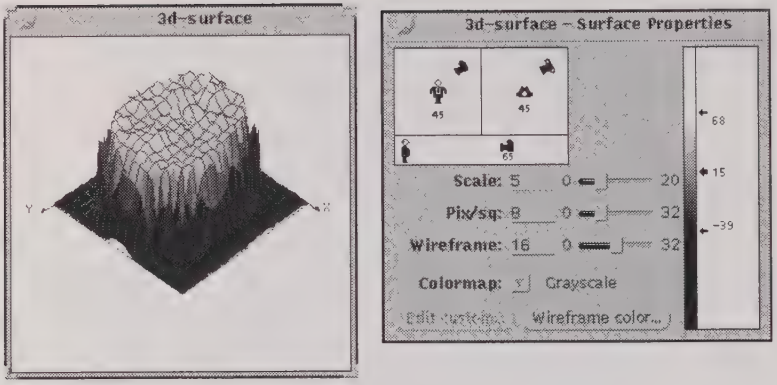

Figure 9.2: Surface Operator Output and Properties

9.2.1 Properties Panel

In addition to the usual zoom and pan controls, the **surface** properties panel provides you with 3-D viewing controls (§10.19) and these controls:

Scale The magnification applied to the image intensity to compute the height of each point on the surface. A smaller magnification means the surface will appear flatter.

Pix/sq Pixel per square. The 3-D surface is drawn as a rectangular mesh. Each rectangle (square) on the surface represents an area of one or more pixels in the image. A small **Pix/sq** gives you a fine and more accurate surface, but takes longer to render. So when you are adjusting your view angle, distance, etc, it is a good idea to first set **Pix/sq** to a large value, do the viewing adjustments, and then set **Pix/sq** to the value you want.
Pix/sq behaves slightly differently on a color and on a monochrome display. On a color display, no boundaries are drawn between the squares. Boundaries are drawn with the size specified by **Wireframe** (see below). **Wireframe** is forced to be always larger than or equal to **Pix/sq**.
On a monochrome display, boundaries are drawn between the **Pix/sq** squares, and the **Wireframe** is not drawn, *except* when **Pix/sq** is set to zero.
When **Pix/sq** is zero, then for both color and monochrome displays, the 3-D surface will no longer appear solid. Instead, a wireframe is drawn at the size of **Wireframe**. Figure 9.3 shows such a view.

Wireframe Specifies the size of the visible grid overlaid on the 3-D surface. **Wireframe** behaves differently in color and monochrome displays. See **Pix/sq** above for details.

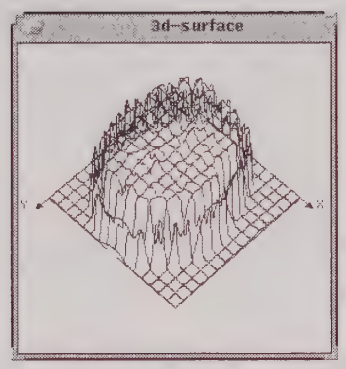

Figure 9.3: 3-D Wireframe

You can control the color map used for the surface from its properties panel, similar to what you do for images (§10.18.2).

9.3 Graph

The **graph** operator accepts a variety of input types and plots the data on a graph. Figure 9.4 shows an example of a multi-data-set plot. Graphs can be printed (§8.5).

9.3.1 Input Types

The input to **graph** can be either one of the following:

- A vector of integers or floating-point numbers: each item in the vector is plotted against its position within the vector.
- A vector of points (ordered pairs): the y value of each point is plotted against its corresponding x value.
- A vector of vectors of integers, floating-point numbers, or points: each vector is plotted as a separate curve within the same graph.

When multiple data sets (curves) are shown, different colors (on color displays) or line styles (on monochrome displays, and when graphs are printed on a monochrome printer) are used to distinguish between the sets. The legend on the bottom identifies the curves.

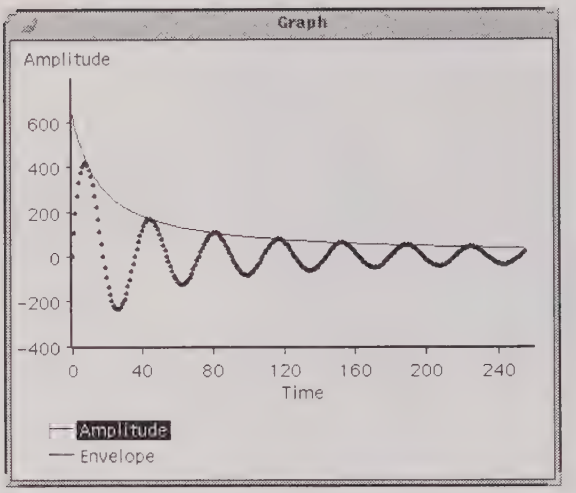

Figure 9.4: Multi-Data-Set Graph

9.3.2 Operator Parameters

The **graph** operator provides these operator parameters which can be set before the operator is executed:

grTitle The title of the graph. If there are multiple **graph** operators in an Igraph, you can send all the plots to the same graph by setting the **grTitle** of the operators to the same name. Also, if you have set **Auto clear** (§10.4.1) to false, then you can plot curves from different Igraphs to the same graph, provided that the **grTitle** are set to the same name.

Name The name of the curve or curve set. If the input is a list of curves, then each individual curve is labeled with this name plus an identification number, e.g. curve-1, curve-2, etc.

xaxis The label on the X-axis.

yaxis The label on the Y-axis.

axisXScale If set to **Auto**, then the range of the X-axis is automatically calculated to just contain all the curves in the graph. The tick marks are calculated so that the labels are nice round numbers. If set to **Manual**, the range specified in the **xaxisRange** parameter is used, regardless of the actual data range of the curves.

xaxisRange The range of the X-axis. Applicable only if **axisXScale** is set to **Manual**. The format must be two numbers separated by a space: **minumum maximum**, Examples: 0 100, -0.01 2.5

axisYScale Similar to **axisXScale**, but applies to the Y-axis.

yaxisRange Similar to **xaxisRange**, but applies to the Y-axis.
type The type of curve (§9.3.4) to plot for the data set.
cumulative If **No**, then every time a data set is sent to the graph, all previous curves are deleted. If **Yes**, the new data set is added to the graph as a new curve. The X and Y axes are recomputed to accommodate all data sets, if the scales are set to **Auto**.
showDataPts Curves plotted as type **Line** (§9.3.4) are plotted by joining data points with line segments. If **showDataPts** is set to **Yes**, then the original data points are shown as dots on the line also. This parameter has no effect on other curve types.

9.3.3 Properties Panel

The **Graph** properties panel (Figure 9.5) has the following controls:

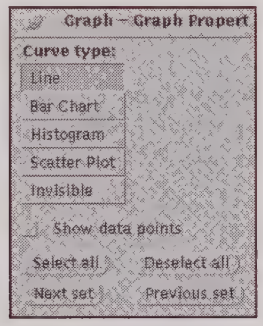

Figure 9.5: Graph Properties Panel

Curve type The type of curve (§9.3.4) to plot the for the selected data sets.
Show data points If checked, and if a curve is plotted as a line, then the actual data points are plotted on top of the line. No effect on other curve types.
Select all Select all curves.
Deselect all Deselect all curves.
Next set Whenever a large number of curves are plotted on the same graph, the legend on the bottom only show a subset of the curves (initially, the first few). Click this button to display the next set of curves on the legend.
Previous set Display the previous set of legends in a multi-curve plot.

9.3.4 Curve Types

Data sets can be plotted in a variety of ways:

- Line
- Histogram
- Bar chart
- Scatter plot
- Invisible

A data set must be selected before its attributes (curve type, e.g. line or bar chart, or whether data points should be shown) can be changed. Data sets are selected by selecting their corresponding legends. A selected legend is shown in reverse. You can also use the **Select all** or **Deselect all** button on the properties panel.

9.3.5 Displaying Exact Coordinates

You can display the coordinates of various positions on the graph: click the left mouse button at the position you want (Figure 9.4), the exact coordinates of the mouse position are shown on the graph. If you drag your mouse on the graph, the coordinates will be continuously updated.

9.3.6 Zooming and Panning

Zooming and panning follows the usual procedures (§8.1). Tick marks on the axes are automaticly recalculated after panning or zooming, but not during. Zooming and panning only applies to the data set area, not the axis labels or legends.

9.3.7 Multiple Curves on the Same Graph

As in displayed images, if two **graph** operators have the same title (§9.3.2), then the same window is used. When a new set of data is sent to a **graph** operator, the behavior depends on the **cumulative** parameter (§9.3.2).

9.4 System

The **system** operator allows you to invoke another program from within WiT. It takes a single **sync** token as input and produce an integer token output. The program to execute and its command line arguments should be specified in the **cmd** parameter as a string, exactly as you would type when

invoking the command from the shell. The program is executed when the input token arrives, and the exit status of the program becomes the output token when the program exits.

In addition to allowing you to execute programs you did not write yourself, **system** can be a useful alternative to invoke your own operators, instead of using the client/server mechanism. You can use operators that have a GUI interface, and they can even be compiled with compilers totally incompatible with the compiler used to compile the WiT libraries.

If the program you want to execute requires more than one input which are produced upstream in WiT, and you want to make sure the program is only executed when all the inputs are available, you can convert those object tokens to **sync** tokens and have the **sync** tokens converge in a junction which leads to the **system** operator.

The output from **system** can be used to synchronize the rest of the Igraph. The operator **rdObjSync** is specifically designed to work with **system**. The program executed by **system** can create WiT objects and save them in files, then the output from **system** can trigger any number of **rdObjSync** operators to read the WiT objects back into the Igraph for further processing by other operators. Figure 9.6 (demo Igraph `systemeg`) shows an example of the use of **system** and **rdObjSync**.

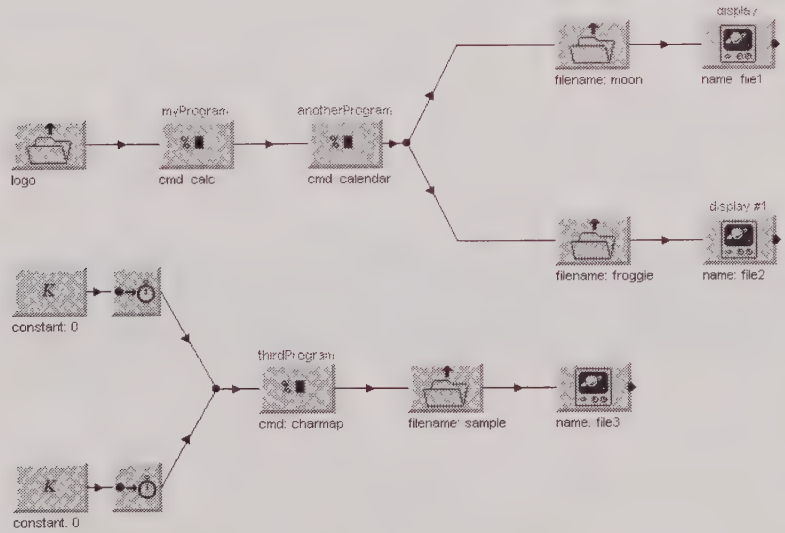

Figure 9.6: Example Use of System Operator

If you want your program to process WiT data objects or produce objects that you can send back to WiT operators, you will need to read and write data in WiT format (see *Programmer's Manual*).

Chapter 10

Parts and Tools List

This chapter describes in detail all the *visible* components of WiT, such as the tool bar, option panels, and various pop-up menus. It does not describe invisible concepts such as how to select objects (Chapter 8, *Techniques*) or initialization files and options (Chapter 13, *Configuration*).

10.1 The Main Window

The main window (Figure 10.1) consists of a menu bar, a tool bar below it, and a large workspace on the bottom. Each item on the menu bar produces a pull-down menu when pressed. Within a menu, items that are followed by an ellipsis (...) bring up a pop-up window when selected.

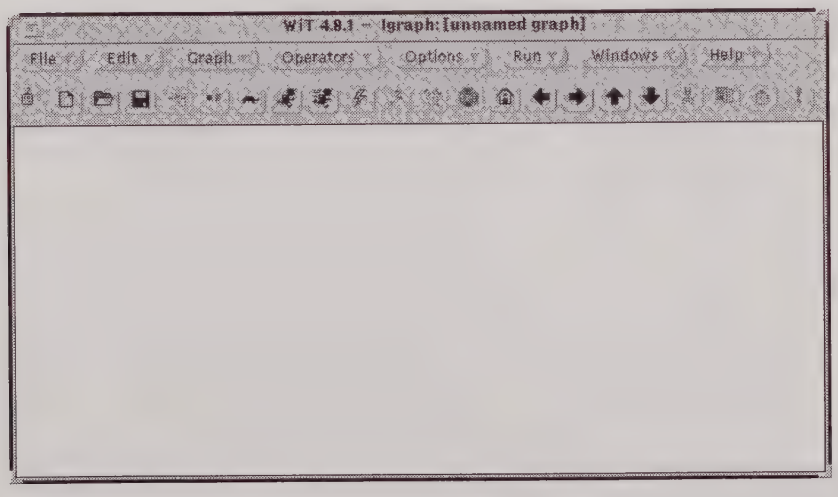

Figure 10.1: WiT Main Window

10.1.1 Main Window Menu Bar

File	**Open...**	Load a new Igraph (§6.2.2) or demand history script (§7.3.1). The current Igraph or demand script is cleared.
	Save	Save the current Igraph (§6.2.3) or demand history script (§7.3.2).
	Save as...	Save the current Igraph or demand history script under a different name.
	Delete...	Delete an Igraph or object file.
	New	Clear current Igraph or demand history script.
	View objects...	Display an object in a detached window (§5.1).
	Print...	Make a hardcopy of the workspace and/or displayed objects (§8.5).
	Print setup...	Set up printer configuration, such as paper orientation, destination, etc., without actually printing anything.
	Exit	Exit WiT. A warning message will be issued to confirm this.
Edit	**Cut**	Delete all selected objects.
	Copy	Copy selected objects to buffer.
	Duplicate	Duplicate selected objects (copy *and* paste).
	Paste	Paste objects in buffer to Igraph.
	Undo	Undo the last *edit* change. Other changes such as erase or running an Igraph cannot be undone.
Graph	**Select all**	Select all objects in Igraph workspace.
	Node group properties...	Set properties for all selected nodes.
	Link group properties...	Set properties for all selected links.
	Trace all	Trace data-flow path for selected objects (§10.1.3).
	Trace inputs	Trace data-flow path for inputs of selected objects.
	Trace outputs	Trace data-flow path for outputs of selected objects.
	Make operator...	Make Igraph into an operator (§6.7).
Operators	**Selector**	Text-based operator selector with wild-card capability and access to operators from all libraries (§10.5).
	Dataflow	Flow control operators (must be present).
	Interactive	Operators that run on the GUI (must be present).
	Read/Write	Operators for reading and writing objects.
	Data Manipulation	Operators which manipulate data format and arrangement, e.g. cast, getField.
	Point	Operators that operate on each pixel individually, e.g. invert, subtract.
	Filter	Digital filters, e.g. low-pass, high-pass, 1 and 2-D convolution.
	Morphology	Operators that operate on shapes, e.g. dilate, erode.

	Transform	Transform operators, e.g. FFT, discrete cosine.
	Measurement	Operators that measure various aspects of an image, e.g. statistics, spectrum.
	Segmentation	Operators that classify portions of an image into distinct segments, e.g. collect blobs, local thresholding.
	Pyramids	Multi-resolution imaging, e.g. decimate, burt.
Options	**General...**	General setup options, such as execution mode and heap size tracking (§10.4.1).
	Data driven...	Data-driven options, such as grid spacing and scheduling mode (§10.4.2).
	Demand driven...	Demand-driven options, such as number of objects cards to be maintained in the workspace (§10.4.3).
	Manager...	Server manager window (§11).
	Save config	Save current configuration (§13.1)
Run	**Start**	Start Igraph (§6.2.4) or demand script execution (§7.3.3), clearing all previous states.
	Pause	Pause Igraph execution. Can resume later with **continue**.
	Continue	Continue after a previous pause.
	Stop	Stop Igraph or demand script execution, cannot continue afterwards.
Windows	**Close all pop-ups**	Close all pop-up windows except the status window. Pop-up windows include library panels, object windows, graphs, etc.
	Reset tiles	Reset object frame tiling to begin at upper left corner of screen (§5.11).
Help	**WiT**	WiT User's Manual.
	Programming	WiT Programmer's Manual.
	About...	General information about WiT.

10.1.2 Group Properties

Sometimes it is desirable to disable whole sections of an Igraph, or set a number of links to a thicker width to emphasize a data path. Such collective changes can be done easily from the **Node group properties** (Figure 10.2), and **Link group properties** panel (Figure 10.3), invoked from the **Edit** menu. The attributes are the same for individual operators or links, the only exception is that you can choose the **As is** option to preserve the original attributes.

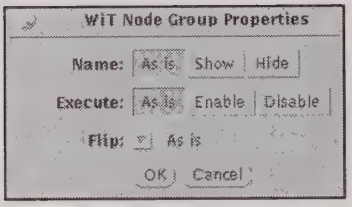

Figure 10.2: Node Group Properties Panel

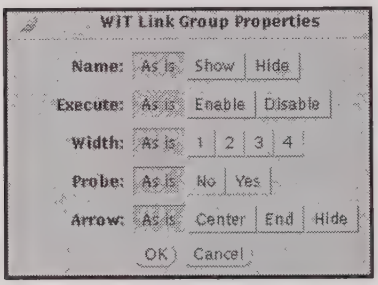
WiT Link Group Properties
Name: As is Show Hide
Execute: As is Enable Disable
Width: As is 1 2 3 4
Probe: As is No Yes
Arrow: As is Center End Hide
OK Cancel

Figure 10.3: Link Group Properties Panel

10.1.3 Tracing

Igraphs tend to get larger and larger. Particularly because WiT can be used to control hardware where Igraph operators can be made to represent small operations in order to provide more graphical programming flexibility. When Igraphs contain hundreds of operators and links, with parallel data-flow, it can become difficult to follow how data will travel on the graph without actually running it. The trace utility from the **Graph** menu provides a convenient way of studying data paths in a large Igraph.

For example, in an Igraph with a large cross-bar switch, it is useful to be able to select some links or operators and trace what other operators are related in terms of data-flow. Tracing even works across sub-graphs. So you can select an operator at the top level Igraph, which contains hierarchical operators. When you trace the operator, and if the data path related to that operator involves any of the sub-graphs, at any level, all such items, regardless of whether the sub-graphs are currently displayed or not, will be put on the selected list (and therefore highlighted).

In wiring cross-bars, it is desirable to use the tracing function to locate which ports of the crossbar are to be connected. However, if you start the connection by double clicking at one of the ports, all selected objects (i.e. traced objects) become dehighlighted, so the port that you want to connect to is no longer highlighted. You can easily avoid this problem by using '.' on the keyboard to start and terminate the wiring (§6.4).

Tracing Inputs

When tracing inputs, if an operator is fire-on-all (§12.3.2), then all its inputs are traced.

If an operator is fire-on-any, and if there is only one of the inputs is connected, then that input is traced. Otherwise, none of the inputs are traced.

For hierarchical operators, the underlying graph structure is used to trace only those inputs related to the selected (already traced) objects.

Tracing Outputs

When tracing outputs, if an operator is fire-on-all (§12.3.2), then all its inputs must be traced before its outputs are traced. If an operator is fire-on-any, then all its outputs are traced if *any* one of its inputs is traced.

Once a primitive operator is added to the traced list, all its outputs are traced.

For hierarchical operators, the underlying graph structure is used to trace only those outputs related to the selected (already traced) objects.

10.1.4 Tool Bar

The tool bar provides a set of push buttons to perform many of the commonly used operations. and an indicator for the current execution state of WiT.

Tool Bar Status Indicator

	Execution stopped, waiting for user action: edit Igraph, start run, etc.
	Executing at **walk** speed.
	Executing at **run** speed.
	Executing at **sprint** speed.
	Executing at **warp** speed.
	Waiting for user input for an operator.
	User has requested execution to be stopped.
	User has requested execution to be paused.
	Execution has been successfully paused.

Tool Bar Buttons

Tool bar buttons provide short cuts to do commonly needed tasks. All the functions of the buttons can also be performed with the pull-down menus. The equivalent menu procedures for each tool bar button are listed below:

Erase current Igraph or demand script (**File/New**).
Read Igraph or demand script (**File/Open...**).
Save Igraph or demand script (**File/Save as...**).
Switch to data-driven mode (**Options/Setup...**, **Data-driven** for **Run mode**).
Switch to demand-driven mode (**Options/Setup...**, **Demand-driven** for **Run mode**).
Run at **walk** speed (**Options/Setup...**, **Walk** for **Speed**, then **Run/Start**).
Run at **run** speed (**Options/Setup...**, **Run** for **Speed**, then **Run/Start**).
Run at **sprint** speed (**Options/Setup...**, **Sprint** for **Speed**, then **Run/Start**).
Run at **warp** speed (**Options/Setup...**, **Warp** for **Speed**, then **Run/Start**).
Pause execution (**Run/Pause**).
Continue execution after pause (**Run/Continue**).
Stop execution (**Run/Stop**).
Make hierarchical operator for current Igraph (**Graph/Make operator**).
Home workspace to upper left corner (**Zoom/Pan...** item from pop-up menu).
Pan workspace left (**Zoom/Pan...** item from pop-up menu).
Pan workspace right (**Zoom/Pan...** item from pop-up menu).
Pan workspace up (**Zoom/Pan...** item from pop-up menu).
Pan workspace down (**Zoom/Pan...** item from pop-up menu).
Cut selected Igraph objects or first demand card (**Edit/Cut**).
Copy selected Igraph objects (**Edit/Copy**).
Paste selected Igraph objects (**Edit/Paste**).
Undo last Igraph editing action (**Edit/Undo**).

10.2 Status Window

The status window is used to report messages and warnings. It is resizable and can be hidden and redisplayed using the **Show status** check box in the Setup panel (§10.4.1). Messages and warnings are still written to the status window even if it is not displayed, so that you can examine the messages later. In addition to writing messages to the status window, you can also tell WiT to write the same messages out to a file, using the **Log status to file** check box in the Setup panel. This log file is closed when you uncheck the check box. So be sure to uncheck this box before you examine the contents of the file.

10.3 Workspace Menus

10.3.1 Data-Driven Mode Workspace Menu

Select Select workspace for printing (§8.5)

Cut	Delete selected objects.
Copy	Make a copy of selected objects.
Duplicate	Copy *and* select objects and start pasting.
Paste	Paste copied objects.
Undo	Undo the last edit action.
View all	Set the workspace scale so that all objects are visible.
Zoom/Pan...	Bring up a zoom/pan control panel (§10.11).
Normal size	Reset the workspace to original scale.
Redisplay	Redisplay the workspace.

10.3.2 Demand-Driven Mode Workspace Menu

Select	Select workspace for printing (§8.5)
Cut	Delete the first object.
Duplicate	Make a copy of the first object, and push all the objects down one slot.
Clear	Delete all objects.
View all	Set the workspace scale so that all cards are visible.
Zoom/Pan...	Bring up a zoom/pan control panel (§10.11).
Normal size	Reset the workspace to original scale.
Redisplay	Redisplay the workspace.

10.4 Options Panels

10.4.1 Setup Panel

The Setup panel provides general setup options (Figure 10.4). The entries are:

Run mode Select between demand or data-driven mode.

Object placement This option controls how new object (image or text) windows are to be placed:

- Random — random locations on the screen.
- Mouse — current mouse position.
- Tile — objects are tiled (i.e. without overlap) across the screen.

Double click Set the interval in milliseconds between clicks of a mouse button before the two clicks are recognized as one double-click event rather than two separate clicks.

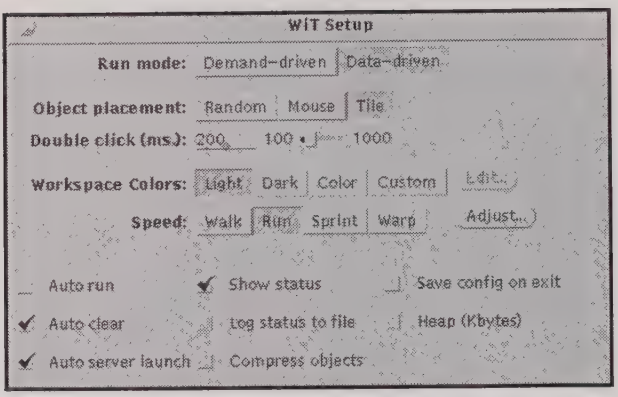

Figure 10.4: Setup Panel for General Options

Workspace Colors This option controls colors used operators, links, etc:

- Color — multi-colored.
- Light — dark objects on a bright background.
- Dark — bright objects on a dark background.
- Custom — (color displays only) user defined color scheme.

Speed The speed control determines whether how much feedback WiT provides when an Igraph is run or when a demand script is played back. **Walk** is slowest and **warp** is fastest. See Section 6.2.4 and Section 7.3.3 for details.

Adjust... This will bring up a new panel for you to adjust the delay associated with each execution speed. The delay affects minimum operator execution time and token animation time in data-driven mode and script playback speed in demand-driven mode. Since some computers are faster than others, the numbers should be treated as relative rather than absolute (such as number of seconds). In other words, adjust the numbers until the delay is satisfactory, without paying much attention to what the delay value actually is.

Auto run If checked, whenever an Igraph or demand history file is loaded (including on WiT startup), it is automatically run (§13.1.1).

Auto clear If checked, pop-up object windows are automatically taken down when a Igraph or demand history is run (§13.1.1). Turning off **Auto clear** is useful when you want to compare results from different Igraphs.

Auto server launch If checked, servers specified in the configuration file (§13.3) are automatically launched (§13.1.1).

Show status If checked, the WiT status window (§10.2) is displayed. If not checked, the status window is hidden.

Log status to file If checked, contents of the status are recorded on a file of your choice (§10.2).

Compress objects If checked, WiT data objects are compressed when written to files (§13.1.1).

Save config on exit If checked, the current WiT configuration is saved to the initialization file (§13.1.3) when you exit WiT.

Heap If checked, the amount of data memory (the size of the heap, in Kbytes) currently in use by WiT is reported next to this check box.

10.4.2 Data-Driven Options

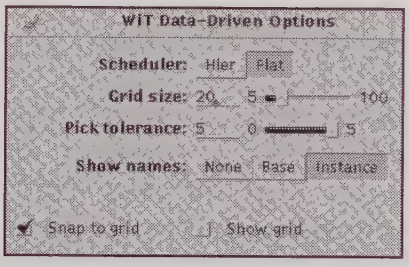

Figure 10.5: Data-Driven Options Panel

Scheduler Select the method that WiT uses to choose which operator to run next (§12.3).

Grid size Set the desired grid size in pixels. In addition to improving readability, the grid is also used for snapping icons and links.

Pick tolerance Specify the accuracy (in pixels) by which the user may pick an Igraph object such as a link or icon. The tolerance relates to how close in screen pixels the mouse arrow must be to an object before it is picked. Closely spaced icons, for example, require a smaller pick tolerance. The Igraph cursor will change to reflect the pick tolerance. Pick tolerance is *not* affected by settings of **Snap** or **Grid**.

Show Names Controls how Igraph object instance names are displayed (§6.6.8).

Snap to grid If checked, Igraph objects (icons or links) placed on the workspace will be snapped to the nearest grid point. Turning snap off will generally make it extremely difficult to create a neat looking graph. It is probably better to reduce the grid size rather than turn off snap when you are trying to fine position objects.

Show grid If checked, a grid is shown on Igraphs (but not on demand-driven workspace). Snapping is not affected by grid visibility.

10.4.3 Demand-Driven Options

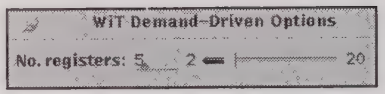

Figure 10.6: Demand-Driven Options Panel

No. registers Set the number of registers that are used when operating in demand-driven mode. More registers give you more flexibility in data manipulation, but require more memory for storage. This is an important consideration if your computer has a small amount of memory, particularly when large images are manipulated.

10.5 Operator Selector

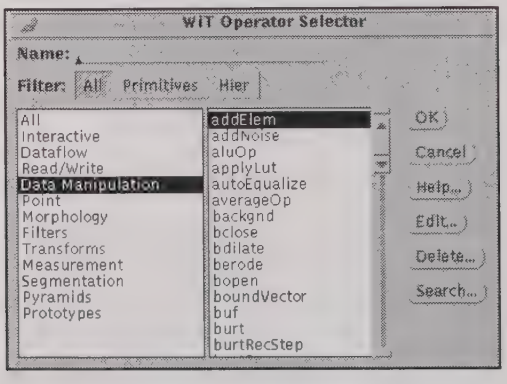

Figure 10.7: Operator Selector Panel

This panel provides many useful ways to select an operator:

- On the left list, you can select a library you are interested in, or you can choose to look at *all* libraries at the same time. Once a choice is made by clicking the mouse at the appropriate item on this list, the corresponding list of operators is shown on the right list.
- Double-clicking on an item on the right list launches the operator, or you can select an operator first by clicking once and then click the **OK** button.

- Regardless of the library selection, you can type the name of the operator you want on the **Name** line. You can use the '*' wild-card on the **Name** line. Operators within the library currently selected which match the name typed are shown in the right list.

The **Filter** item allows you to display only primitive or hierarchical operators (§6.7).

The functions of the buttons on the right hand side are:

OK	Launch operator in **Name** field.
Cancel	Bring down window without launching any operator.
Help...	On-line help for operator.
Edit...	Edit the icon (§10.15).
Delete...	Delete the operator from WiT, use caution!
Search...	Do a keyword search to find a relevant operator (§10.6).

10.6 Operator Keyword Search Panel

The keyword search panel (Figure 10.8) is invoked from the **Search...** button on the Operator Selector (§10.5). It is useful when you know what you want to do, e.g. perform edge detection on a image, but do not know what operators are relevant.

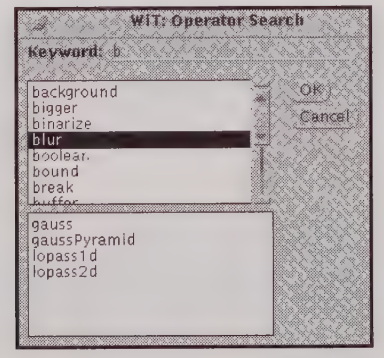

Figure 10.8: Operator Keyword Search Panel

The keyword search panel consists of these fields:

- A **Keyword** field where you type a keyword.
- A **Keyword List** window, where matching keywords are displayed as you type your keyword.
- An **Operator List** window, for showing relevant operators when you select a keyword from the **Keyword List**.

To perform a keyword search:

1. Click the **Search...** button on the Operator Selector. The keyword search window comes up.
2. On the **Keyword** field, type the keyword you have in mind, e.g. edge. The keyword list below is updated as you type.
3. When you can see the name on the keyword list, click the left mouse button on the item. A list of relevant operators is then listed on the bottom window.
4. If the keyword list becomes empty before you finished typing your keyword, it means there is no such keyword. Try to phrase your idea differently, e.g. using *detect* instead of *edge*.
5. Select the operator name you think is most appropriate, then click the **OK** button. The keyword search window goes down and the operator name is copied to the **Name** field of the Operator Selector.
6. You can now invoke the operator (§4.1.2) or get help about it (§4.4).

10.7 File Panel

The **File** panel (Figure 10.9) is used whenever the user is required to select a file, such as loading and saving Igraphs, viewing objects, and deleting files.

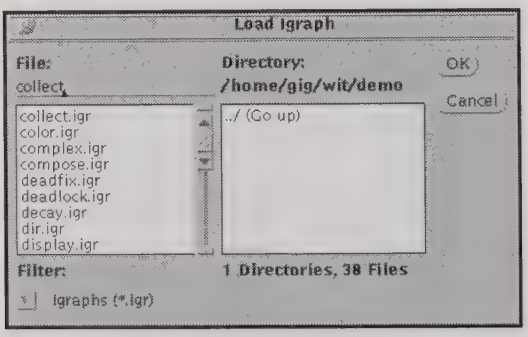

Figure 10.9: File Panel

Depending on what operation you are performing, the **File** panel will list the *appropriate* files in the current directory. For example, if the **File** panel is brought up with the **View...** item from the main panel **File** button, the **File** panel lists all the WiT objects (files with a '.wit' extension) in the current directory, but nothing else. If the **File** panel is invoked with the **Load...** item, then **File** panel will list only the Igraphs (files with a '.igr' extension) in the working directory.

Regardless of the context from which the **File** panel is invoked, you react with it in exactly the same way. You can type the file name you want on the space provided on the top. The current directory

is always shown below this field. You can also select from the list of files displayed (presented as a **Choice List**) with your mouse. After you make your selection (or finished typing), click the **Apply** button to load the file, or you can double-click at your selection to launch it immediately. The **Cancel** button clears the file name and take down the **File** panel.

The **../ (Go up)** item on the choice list moves you one level up the directory tree. Notice the **Current Directory** item changes as you change directory.

10.8 Choice Lists

In addition to the basic OpenLook input gadgets, WiT uses a **Choice List** gadget to provide better user control when choosing among a list of choices. For example, the **Choice List** is used in the **File** panel and **Font** panel. As an example, Figure 10.9 shows the **Choice List** used in the **File Selection** panel for choosing among file names.

In addition to providing a scroll bar for panning a long list (which is a standard OpenLook gadget), the WiT **Choice List** interactively highlights the item that your mouse is currently pointing at, and when you click at an item, a check mark appears to the left of the item to indicate that it has been selected. When you then hit the **Apply** button, the selected item is applied to the current operation. For example, if you are using the **File** panel to read an Igraph, then clicking **Apply** button causes the Igraph to be read. You can also speed up the selection-then-apply procedure by double-clicking at an item, which will cause the item to be applied immediately.

10.9 Font Panel

Whenever the user needs to specify a new font for WiT to use, WiT brings up a **Font** panel (Figure 10.10).

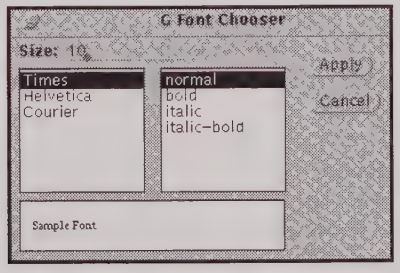

Figure 10.10: Font Panel

Use of this panel is self-explanatory. The **Size** field lets you type in the font size. Units are in points, but most likely you will quickly learn what size to use for different purposes, so the size serves primarily as a relative rather than an absolute quantity, therefore the units used is unimportant.

Below the **Size** field, there are two list windows, the one on the left for the font family, such as 'Times' or 'Helvetica'. The one on the right is for font style, such as 'roman' or 'italics'. Any of the size, family, and style choices can be mixed and matched at will.

As with most WiT panels, you have to click **Apply** on the panel in order for your selection to take effect.

10.10 Color Panel

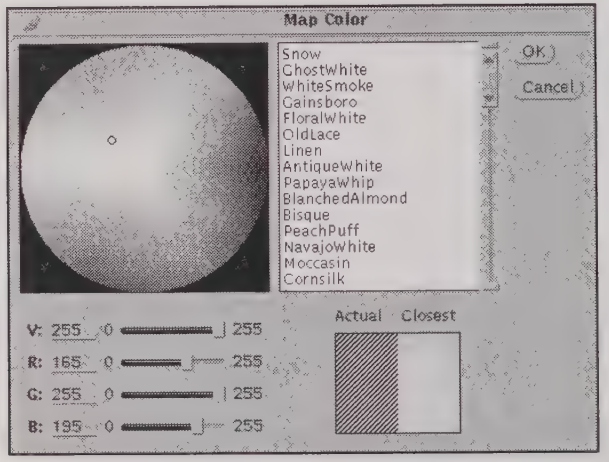

Figure 10.11: WiT Color Panel

Whenever the user needs to specify a new color for WiT to use, WiT brings up a **Color** panel (Figure 10.11). The **Color** panel presents different methods of specifying a color, because different people have different preferences, and the most appropriate method also varies depending on the application. The color disc on the upper left corner (based on the HSV color model, which stands for Hue, Saturation, Value) is useful when you want to specify a color approximately. As you drag the mouse in this area, the color under the mouse is displayed in a rectangular box on the lower right corner. This box is divided into two parts, the actual (exact) color on the left, and the closest color that WiT can provide on the right. The closest color cannot always be exact because WiT only uses a limited number of colors from the window manager. The **R**, **G**, and **B** sliders on the lower left are also updated as you drag the mouse inside the color disc. These sliders represent the red, green, and blue components of the selected color.

RGB is another method of specifying colors. People who are used to mixing paints may find that RGB provides more intuitive control. Even though mixing light and mixing paint work in opposite ways, the procedure of adding small amounts of primary colors to achieve a desired color can easily be mastered by someone with an understanding of color. For example, to get a specific shade of brown, we can start with gray, add a little red, adjust with a little green (which when mixed with red *light*, gives a shade of yellow). Proceeding this way, it is not difficult to achieve the *exact* shade of

brown one wants. As you change either one of the R, G or B components, the crosshair on the color disc moves to reflect the corresponding HSV representation of the same color.

The color disc only allows you to select the hue and saturation for the HSV model. The value component can be altered with the **V** slider above the RGB sliders. A high V means bright colors, and a low V means dark colors.

In addition to RGB and HSV, the **Color** panel also provides a list of common color names on the upper right. You can select these colors by clicking with the left mouse button. Again, both the RGB and HSV input gadgets are adjusted accordingly to reflect the selected color.

Whichever input method you employ, you must click the **OK** button to activate the new colors.

10.11 Zoom/Pan Panel

In addition to using combinations of mouse buttons and motion to do zoom and pan (Section 8.1), WiT sometimes also offers a **Zoom/pan** panel (Figure 10.12) as an alternative. For example, zoom/pan on the main WiT workspace can be performed using the mouse or by bringing up the **Zoom/pan** panel from the workspace pop-up menu.

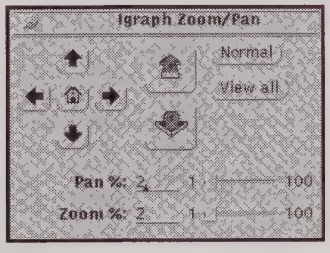

Figure 10.12: Zoom/pan Panel

The arrow buttons on the left controls panning up, down, left, and right. Every click of the button pans the view area in the indicated direction a certain amount, which is controlled by the **Pan %** slider below these buttons. The amount is specified in percentage of the visible area. The center button bring the viewing area 'home.' For example, when using this panel with an Igraph, 'home' positions the viewing area such that the upper-left most object in the Igraph is positioned at the upper-left corner of the workspace.

The magnifying glass buttons in the middle of the **Zoom/pan** panel control zooming. The zoom amount is controlled by the **Zoom %** below these buttons. Again, as in panning, the amount is specified in percentage of the visible area. The **Normal** rescales such that objects appear in their original size, and **View all** scales the view area so that all objects are visible.

10.12 Demand-Driven Operator Properties

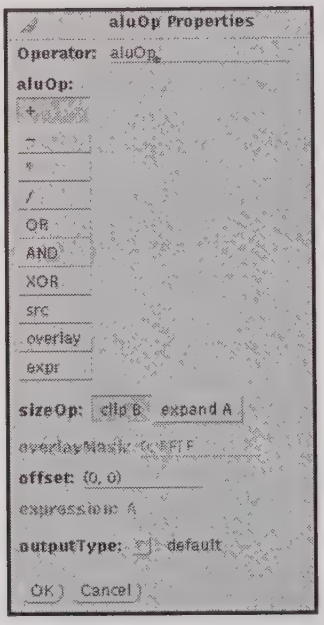

Figure 10.13: Example Demand-Driven Operator Properties

The parameters **width**, **height**, **type**, **fillType** and **fillval** in Figure 10.13 are specific to this operator. The standard properties are:

Operator Name to be used on the labels of cards for any output data objects. You can change it to any string you want, including names with spaces (§7.2.2).

OK Apply properties and take down the properties panel. The operator is then executed.

Cancel Cancel all data entered and take down the properties panel. The operator is *not* executed.

10.12.1 Data-Driven Operator Properties

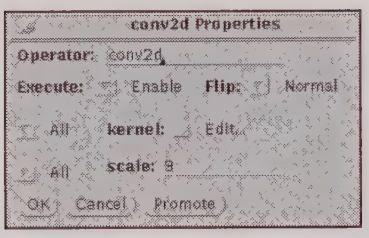

Figure 10.14: Example Data-Driven Operator Properties

Operator By default, when you create an operator instance on an Igraph, the first instance of that operator will have the same name as the name of the operator shown on the library. We will refer to the name of the operator on the library as the *operator name*, and the name of the operator on the Igraph as the *instance name*. If you have more than one instance of one operator in an Igraph, subsequent instances will have a number appended to the operator name, e.g. **rdObj #1**, **rdObj #2**, etc. The **Operator** control allows you to override this default behavior. For example, you may want to name some **rdObj** operators as **readXRayImage**, **readProfileData**, etc., so that the Igraph is easier to understand, and you can refer to a particular instance of **rdObj** in your documentation (§6.6.8).

Execute **Disabled** means that although the operator is shown on an Igraph, it will not be executed. For example, you may have an Igraph with multiple starting nodes. By disabling a starting node, you can disable a whole branch of an Igraph. A disabled operator functions as though it is not on the Igraph at all. Tokens sent to a disabled operator will stop there. Disabled operators are shown with a cross-hatch pattern.

Bypass means the operator performs no function, but allows tokens to pass through. For example, you may have a low-pass filter in an Igraph, and you want to see what the result will be if the filter is taken out. Only operators which have the same number of inputs and outputs *and* the input and output *types* have a one-to-one correspondence can be bypassed. Bypassed operators are shown with a large arrow superimposed on the icon.

Flip Flip the icon in the X or Y direction, or both. This is useful when you have loop-backs or branches coming from the top of an icon. Input and output ports are flipped together with the icon graphics.

OK Apply properties and take down the properties panel.

Cancel Cancel all data entered and take down the properties panel.

Promote Promote a parameter for use in a hierarchical operator.

Expand Appears on hierarchical operators (§6.7) only. Clicking on this button will bring up the Igraph represented by a hierarchical operator in a detached window.

For each operator parameter a choice control is presented on the left which allows you to control whether a parameter is displayed with its name and value (**All**), value only (**Value**), or not displayed at all (**None**). If **Input** is selected, the parameter will be fed by another operator as an input. The value field for that parameter will be disabled, and a port will appear on the bottom of the operator (§6.3.5).

10.13 Link Properties

Name Change the name of a link. See Section 6.6.8 for the significance of object names.

Execute Enable/disable a link. Disabled links are shown as a dashed line. No data flows along a disabled link.

Width Set how thick a link should be drawn. This can be useful if you want to emphasize certain links. Link thickness does not affect Igraph execution in any way.

Probe If **Yes** is selected, a probe (§6.4.6) is placed at where you clicked your mouse to bring up the properties panel. If you want to move the probe, you have to select **No** first and **OK** the properties. Then click the right mouse button on the position of the link that you want the probe to be, and select **Yes** for **Probe**.

OK Apply properties and take down the properties panel.

Cancel Cancel all data entered and take down the properties panel.

Promote Promote a parameter for use in a hierarchical operator.

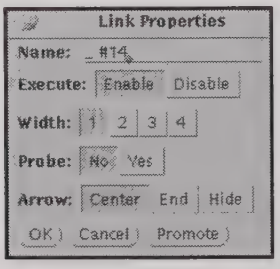

Figure 10.15: Link Properties Panel

10.14 Array Editor

When WiT requires the user to input two-dimensional data, such as convolution kernels, it brings up an array editor (Figure 10.16). The editor consists of three regions: a menu bar, a panel, and the actual array.

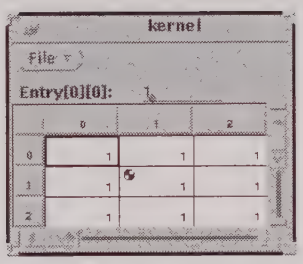

Figure 10.16: Array Editor

10.14.1 Menu bar

The menu bar contains a single item **File**, which gives you the following functions:

Load...	Load a data file into the array.
Save	Save array data to the file the data came from originally.
Save as...	Save array data to new file.
Done	Apply contents of editor and take it down.
Size...	Change number of rows and columns in array.

10.14.2 Panel

The array editor panel contains a single control called **Entry**. It shows the value of the selected array element (see below). Changes to an array element must be performed here. The selected item is highlighted with a bold line box in the array region.

10.14.3 Array

The array region consists of three areas: row labels, column labels, and the array data. At any one time, one of the array items is the selected item, and it is highlighted with a bold line box. The value of this entry also appears in the **Entry** item on the array editor panel. When you modify the **Entry** item value, the new value is immediately copied to the actual array item. To change the selected item to a different element of the array, click the *left* mouse button on the desired element. You can also *copy* the value in the **Entry** item to any element by clicking the *middle* mouse button at the desired element.

For some applications, it is necessary to define an 'origin' in the array. This is indicated by a ◉ symbol on the upper right corner of the array element. You can define the origin at a new position with the *right* mouse button.

When the array size is larger than the window size, scroll bars will appear. You can use these scroll bars to look at different regions of the array. You can increase or decrease the visible portion of the array by resizing the window itself. Notice that the actual size (rows and columns) of the array is not affected by changes in the window size.

10.15 Icon Editor

WiT has a built-in icon editor (Figure 10.17) which not only lets you design the appearance of operator icons, but also allows you to move input and output ports, among other things. Because it is built-in to WiT, when you change an icon with the icon editor, those changes are updated immediately on the Igraphs you are looking at.

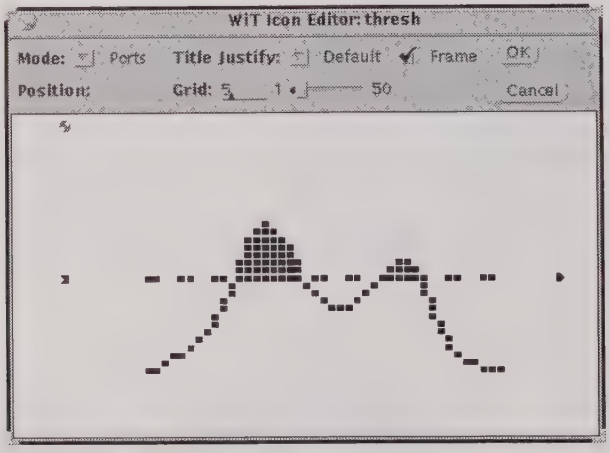

Figure 10.17: Example Icon Editor Session

There are two ways to invoke the icon editor:

- Point the mouse at an operator in a graphical library panel, click the right mouse button to bring up the pop-up menu, and select the **Edit** item.
- Select an operator from the Operator Selector, and click the **Edit** button in the **Operator Selector** panel.

The icon editor has these controls:

Mode	Input mode, see below.
Title Justify	Specifies how the operator title text is justified.
Position	Position of mouse in icon coordinates.

Grid Size of grid in icon coordinates. Ports, origin, and title in the icon editor always snap to this grid (not the same as the Igraph grid).

Frame If checked, the icon will be drawn with a frame. On a color display, framed icons have a 3-D look.

The icon editor operator works in four possible input modes, selectable with the **Mode** item as follows:

Ports Move input or output ports.

Icon Change icon graphics. A click of the left mouse button sets a pixel, the middle button clears it.

Origin Move the origin (hot spot) of the icon. The origin is indicated as a ◉ symbol. When an operator icon is placed on the Igraph with **snap** turned on, the origin of the icon is snapped to the closest grid intersection point. You should place the origin in such a way that the input and output ports will all be lying on the same grid as the origin.

Title The title of an operator is normally shown on the top of the operator icon, centered. You can change the position and alignment of the operator title text in this mode. The title position is shown as a ▸ symbol.

10.16 Operator Library Pop-up Menu

Edit icon Bring up the WiT icon editor (§10.15). for modifying the icon graphics or port positions of the icon.

Delete Delete the icon from the library. Only the icon is deleted. The actual program code (if the operator is primitive) or underlying Igraph (if the operator is hierarchical) is retained.

Help Bring up on-line help for the operator.

10.17 Image Pop-up Menu

Select Select the image for later printing.

Properties... Bring up image properties panel.

View all Display image in its entirety, sized to the shape of the window. Useful after panning or zooming.

10.18 Image Properties

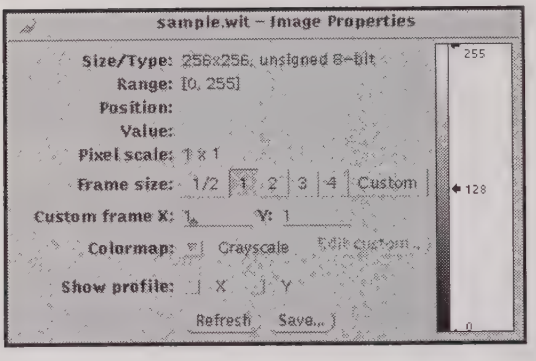

Figure 10.18: Image Properties Panel

Size/Type	Original width and height of image, and the type of each pixel.
Range	The actual pixel value range within the image.
Position	Position of mouse in image coordinates. Upper left corner is (0,0).
Value	Value of pixel at which mouse is pointing at.
Pixel scale	Screen size of each image pixel.
Frame size	The size of the image window relative to the original size of the image. If **Custom** is selected, the sizes specified in the **Custom frame** explained below is applicable.
Custom frame	The size of the image *window*, or frame, expressed as a floating-point number for the X (horizontal) and Y (vertical) directions, relative to the original size of the *image*. These numbers track changes caused by the built-in choices from the **Frame size** item above, the resize corners on the image window, or zooming in or out with the rubber band area select box (see below).
Show profile	Show a plot of pixel values along a line on the image in the X or Y directions. The plot will be shown in a separate window, initially placed beneath (X) or to the left (Y) of the image. You can move and resize the profile window once it has come up. As you drag (move with left button held down) the mouse up and down on the image, the profile plot will be updated accordingly (§5.2.2). Not available for color images.
Image index	Applicable to vector of images only. Select which image in the vector to display.

Load all Applicable to vector of images only. A vector of images is cached on the display server so that the scroll bar can be used effectively to quickly select different images within the vector. Normally this cache operate like most caches: an image is placed in the cache only when it has been accessed, i.e., when it is displayed. This can create some annoying initial delay when using the scroll bar. The **Load all** button allows you to force all the images in the vector to be loaded in the cache, provided that the cache size is not exceeded. No harm is done if the cache is not large enough. In that case the cache will be filled to capacity.
Consideration must be given to memory consumption. Images typically take up a large amount of memory. Loading a vector of images can cause severe performance degradation to WiT and other programs, due to excessive swapping of RAM data to and from disk. If you do not need to access all components of the image vector, then it may be better to let the cache load the images on demand.
The cache is flushed whenever the image scale has changed, such as when you scale an image using the resize corners. If you wish to pre-load all images in the cache, then you will have to click the **Load all** button every time you resize the image window. The demo Igraph `collect` demonstrates the use of image vectors.

Colormap Behavior of this item is different for grayscale and color images.
For grayscale images, three colormap types are available:

- Grayscale: Consists of different shades of gray. The number of shades can be user configured (§13.2).
- Pseudo: Grayscale values in an image are mapped to a set of colors representing the visible light spectrum from blue to yellow to red. Pseudo-color is useful for enhancing the contrast between different parts of a grayscale image.
- Custom: Bring up a **Custom Color** panel which allows you to specify colors for specific pixel intensity ranges (§10.18.2).

For color images, if the display supports only 256 colors (§13.2), two colormap types are available:

- Shared: A low quality map shared by all displayed color images. The low quality allows all color images to be displayed reasonably at the same time.
- Render: A special colormap is computed for the current image. This map is the best colormap that can be used given the total number of colors available to WiT, but may cause other color images to be displayed with incorrect colors.

Refresh Redisplay the image. Useful when parts of the image are not displayed properly due to actions from other windows.

Save... Save the current image as a WiT object to a file. A filer dialog will come up. The saved object can be read by the **rdObj** operator. See Section 13.8 for more information about which directory the image will be written to.

Note that different image types may have specific properties shown only with those types, e.g. profiles are not applicable to color images, and image index is only applicable to an image vector.

10.18.1 Contrast Ramp Control

In addition to the control gadgets, the grayscale ramp on the right hand side of the properties panel (Figure 10.18) is used to adjust contrast on the image, to enhance certain features for viewing purposes. More technically speaking, it is used to change the size and position of the pixel intensity window through which we look at an image.

The current color map of the image is represented by the ramp. The arrows with numbers on the right indicate the current intensity window settings. You can drag these arrows with the left mouse button to define a smaller or larger intensity window. When the window size changes, the grayscale ramp is remapped to the new range. You can keep the intensity window size fixed but move its position on the gray scale by dragging the middle arrow. The ramp is applicable to both grayscale and pseudo-color colormaps.

10.18.2 Image Colormaps

There are three colormaps you can choose for grayscale images: grayscale, pseudo, and custom. 'Pseudo' maps a grayscale image to a rainbow color map: black (lowest pixel value, 0 for unsigned images) is mapped to violet and white (maximum pixel value) is mapped to red. 'Custom' is also a pseudocolor map, but is user definable.

Custom Image Color Map

WiT allows you to selectively map a range of grayscales to a specified color. When you select **Custom** option for the **Colormap** item, WiT brings up an Image Color Editor window (Fig. 10.19).

If we take the `sample` image as an example, suppose we want to map the background (grayscale range 0-31) to a deep blue, and the range 104-143 to a bright yellow color:

1. On the grayscale ramp on the left side of the **Color Editor**, move the window to (0, 31).
2. Select **Edit/Choose color...** A color panel (§10.10) comes up.
3. Select the blue color you want. E.g. set the RGB value to (0,0,192).
4. Select **Set** from the **Edit** menu. A new entry appears on the box on the right side of **Color Editor**. The entry shows the mapping **(000, 031) (000, 000, 192)**.

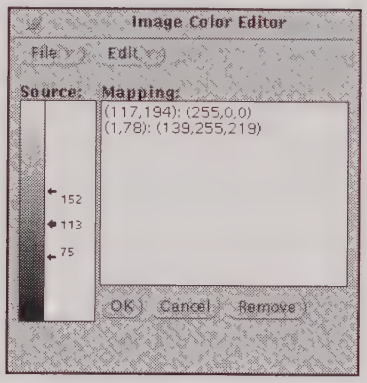

Figure 10.19: Custom Image Color Panel

5. Repeat the process for the range (104-143) on the grayscale ramp.
6. Click **OK**. The `sample` image is now shown with a blue background with some yellow patches. This custom color map will be used for this image for as long as it is displayed.

To delete a color mapping for a particular range:

1. Select (with left mouse button) an entry in the mapping window.
2. Select **Delete** from the **Edit** menu.

You can restore the original grayscale color map by clicking the **Remove** button.

Normally, as you adjust the grayscale ramp, the corresponding pixels in the image is automatically updated with whatever the current selected color is, so that you can see what the effect of the new color will be. But this color is not *installed* yet until you select **Edit/Set**. If you want to previs what the *installed* colors look like, select **Edit/Hide current range**.

The **File** menu allows you to save the mapping to a file and to read in a previously saved map file to apply to the current image.

10.19 Three-Dimension Viewing Controls

When a 3-D object is displayed, such as a surface plot of an image, it is often necessary to change the viewpoint to visualize the object. WiT provides an intuitive control set for this purpose (Figure 10.20).

Figure 10.21 illustrates some of the terms related to 3-D viewing manipulation.

In our 3-D viewing model, we imagine ourselves to be looking through the view finder of a camera, which we will refer to as the eye (E), at some 3-D object (in this case, a surface that looks like a

Figure 10.20: Three-Dimension Controls

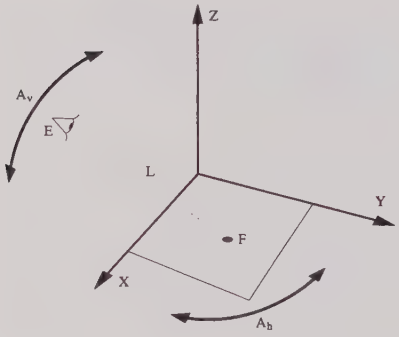

Figure 10.21: Three-Dimension Control Terminology

piece of mountainous land). The focus (F) is the point in 3-D space that we are pointing our camera at. Axes are assigned to the 3-D surface: the X-axis is along the width of the image, Y is along the height of the image, and Z is the intensity of the image, represented as the elevation of the 3-D surface. The line (L) between the eye (E) and the focus (F) is called the view axis.

We always stand upright (in other words, we do not rotate the eye about the view axis). Other than this restriction, we are free to move the eye (but not the focus) to wherever we want. If we imagine ourselves to be standing on the X-Y plane of the 3-D surface we are looking at, and we walk around the focus, then we are varying the angle between L and X (A_h), which we will call the horizontal angle. (We are also varying the angle between L and Y, but that is redundant.) If we step on a ladder to look at the surface from a higher vantage point, then we are varying the angle between L and Z (A_v), this we will call the vertical angle.

The 3-D controls use the symbol of a man as the view object and the symbol of a camera as the eye. There are three controls: the horizontal angle, vertical angle, and distance.

Horizontal Angle The angle between the view axis projected on the X-Y plane and the X-axis (A_h). The horizontal angle is zero when the camera is on the positive X-axis, 90 degrees when the camera is on the positive Y-axis.

Vertical angle The angle between the view axis and the X-Y plane (A_l). When the vertical angle is zero, the camera is at the same level as the man. When the vertical angle is 90 degrees, the camera is looking straight down at the head of the man.

Distance The distance between the camera and the focus, i.e. the length of (L). This has a different effect than the two-dimensional zooming available in most windows. Our camera analogy will help explain the difference.
The view displayed in the window is like the picture taken on film in a camera. Move closer to or away from the object we are taking a picture of is the same as changing **distance**. Two-dimensional zooming is equivalent to enlarging or reducing the picture *after* the object has been photographed. Therefore, varying **distance** changes the perspective, zooming does not.

10.20 Print Panel

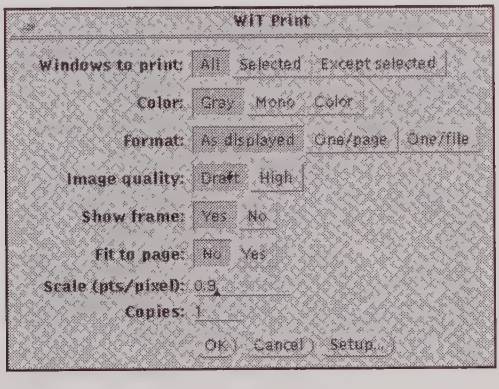

Figure 10.22: Print Panel

Windows to print	**All**	Print all windows.
	Selected	Print only selected windows.
	Except selected	Print only non-selected windows.
Color	**Gray**	Colors are mapped to an appropriate grayscale.
	Mono	Colors are mapped to either black or white. Images are printed the same as **Gray** setting.
	Color	Colors are output with RGB values. Only useful if your PostScript printer has color. More bytes are generated for color images. So select this only if your printer actually outputs in color.

Format	**As displayed**	Frames printed with same relative positions as displayed on screen.
	One/page	Each frame on a separate page, centered.
	One/file	Each frame on a file. The window title becomes the file name.
Image quality	**Draft**	Images are dithered with the resolution of the screen. If you are using a monochrome display, the output will have exactly the same number of dots as you see on the screen.
	High	Grayscale images are output in 256 (8-bit) grayscale. Color images are output in 256 grayscale or 24 bit/pixel format depending on the **Color** control. This allows the printer the freedom to use whatever halftone screen to produce the best results. On low-cost printers, the results may not look very good, but you can send the same file to a professional printer and obtain flawless images. Grayscale images requires much more data to be transferred to the printer, and so may take more time to print, particularly if the link to the printer is serial. The images can be more difficult to photocopy too.
Show frame		Enable or disable a frame with the window name around each printed window (Igraph, images, etc.)
Fit to page		If **Yes**, all windows within one page are scaled to fit the size of the paper. If **Format** is set to **One/page**, then the scale is calulated so that the largest of the windows fits the paper. All pages adopt the same scale.
Scale		If **Fit to page** is **No**, this scale is used to scale windows. A scale of 1 means each pixel on the screen is scaled to 1 printer point (1/72 inch).

Chapter 11

Server Manager

The WiT server manager allows you to configure which GUI/server combination you want to launch, generate operator header files, rebuild servers, etc. In addition to invoking it from within WiT from the **Options/Manager** menu item, you can also start the server manager as a separate program.

WiT manager is used extensively when programming primitive operators for WiT. Features related to primitive operator programming will not be discussed here. Refer to the *Programmer's Guide* for those details.

WiT manager has four areas (Fig. 11.1): a menu bar, a tool bar, a connectivity chart on the left (with a blue background), and a status window on the right.

11.1 Menu Bar

The manager menu bar has two menus:

File	**Open**	Open a configuration file (§13.3).
	Save	Save current configuration file.
	Save As	Save current configuration file as another name.
	Path	Change search paths for Igraphs, objects, servers, and libraries.
	List Objects	List structures of user-defined objects.
	OK	Activate the selected configuration.
	Cancel	Cancel the selected configuration.
Build	**Prototypes**	Generate operator prototype header files.
	Prepare Servers	Prepare all relevant server files, but do not compile.
	Compile Servers	Compile servers.
	Test Operator...	Generate stand-alone test program for an operator.

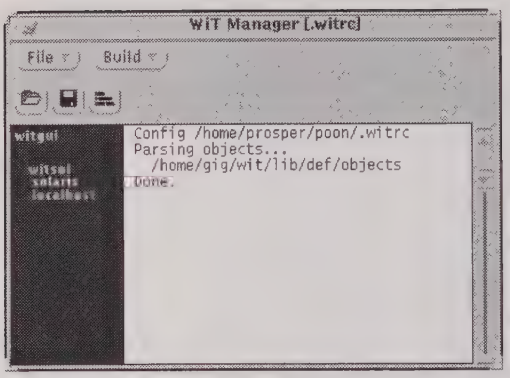

Figure 11.1: WiT Manager

11.2 Tool Bar

The tool bar provides shortcuts to options already available from the menu bar:

Same as File/Open.

Same as File/Save As.

Same as Build/Servers.

11.3 Connectivity Chart

The connectivity chart shows you what servers will be scheduled on which computer. The first line is always **witgui**. You can specify command line arguments to the servers from this window: click the right mouse button on the server name. A property panel comes up (Fig. 11.2).

The controls are:

Shutdown	Name of the server. Can be anything you like.
Type	The type of the operator set supported by this server. Software based servers are usually of type **general**. Servers for specific hardware usually adopt the name of the board, e.g. **p360** for DIPIX P360F, **icvl** for Imaging Technologies ICVL.
Arch	The architecture, **sun4** for SPARC, **win31** for Windows 3.1, **nt** for Windows NT.
Custom	Activated when **Arch** is set to a non-standard type.
IPC	Interprocess communication mechanism. On Windows, you can choose between **DDE** and **socket**. If your computer is not networked, you should choose **DDE**. On UNIX, only **socket** can be used.

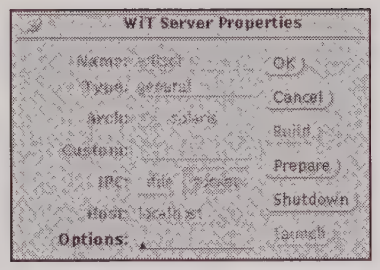

Figure 11.2: Server Property Panel

Host	The computer where the server will be run on. Relevant only when using sockets for IPC. The local machine is **localhost**.
Options	Any other command line options that the server will accept. The set of options may depend on the server type.
OK	Accept the settings and take down the panel.
Cancel	Cancel the settings and take down the panel.
Build	Recompile the server, see **Programmer's Guide**.
Prepare	Prepare the server function stubs, see **Programmer's Guide**.
Shutdown	Shutdown the server. Applicable only if the server is in the current configuration and is running.
Launch	Launch the server. Applicable only if the server has been previously shutdown.

11.4 Status Window

This is an area where WiT manager reports requested information or status messages. You cannot enter anything in this window.

11.5 Choosing a Configuration File

To choose a configuration file:

1. Select the **Options/Servers** menu item. The **Server Manager** window comes up.
2. Do either one of the following:
   - Click on the **Server Manager**.
   - Select the **File/Open** menu item from the **Server Manager**.

   A file dialog comes up.

3. Select the configuration file you want, then click **OK**. The file dialog goes down. The server chart on the **Server Manager** is updated to reflect the new configuration. At this time, the configuration is *not* activated yet. This way you can check whether this configuration is indeed what you want.

4. If the configuration is not what you want, repeat the last step until you are satisfied with the configuration.

5. Select the **File/OK** menu item. The configuration is activated and WiT attempts to reconfigure. Old servers (if any) are shut down and new servers are launched and connected. The operator libraries are updated to represent the new set of operators supported by the new servers.

The active configuration is *always* copied to

```
~/.witrc
```

This copied version is shared by other programs (see the *Programmer's Manual*).

Chapter 12

Advanced Igraph Concepts

WiT adopts the data-flow model of program execution instead of the control-flow model used in most programming languages such as C or BASIC. The main advantage of data-flow is that algorithms can be constructed as intuitive block diagrams and can run in parallel when the computer hardware is capable. However, because most computer languages are control driven, the data-flow model may be foreign to most users. In this chapter, we will take a more analytical look at the data-flow model that WiT uses. This knowledge may be necessary in the design of more advanced Igraphs.

12.1 Brief Review of Data-Flow Concept

In the data-flow model, an algorithm is expressed in terms of operators that process input data *tokens* and produce output data tokens. An operator may be a simple procedure, such as addition or subtraction, or it may be an intricate function consisting of thousands of lines of code. An operator becomes ready for execution when all its input objects are available. With data-flow, it is not necessary to assign names to variables. The relationship of program code (operators) to data (objects that flow along links) is expressed graphically by how operators are connected. This makes the data-flow model more suitable for a graphical environment. Moreover, it is natural to express parallel program execution with data-flow. For example, suppose you want to program the equation $C = (A + B) \times (A - B)$ If you program this in C, you might write

```
T1 = A + B;
T2 = A - B;
C = T1*T2;
```

Notice the use of variable names, particularly `T1` and `T2` for holding temporary results. You may argue that you can program the entire equation in a single C statement and thus avoid temporary variables, but bear in mind that this is a simplified example. The operations + and − may be arbitrarily complex expressions.

Notice also that in reality it is not necessary that the computation of $T2$ should follow $T1$, or vice versa. $T1$ and $T2$ can be computed at the same time, since they both depend only on A and B. Most programming languages do not allow such *parallelism* to be expressed explicitly. Special languages (such as Ada and Occam) are needed. One possible implementation might look like this

```
CoBegin {
  {
    T1 = A + B;
  }
  {
    T2 = A - B;
  }
}
C = T1*T2;
```

where we assumed a hypothetical programming language which has the keyword `CoBegin` which allows you to explicitly specify what instructions can be computed in parallel.

In contrast, this equation can be elegantly expressed in a data-flow diagram (generally called a data dependency graph) as in Figure 12.1. The parallelism is implied by the link branches after the

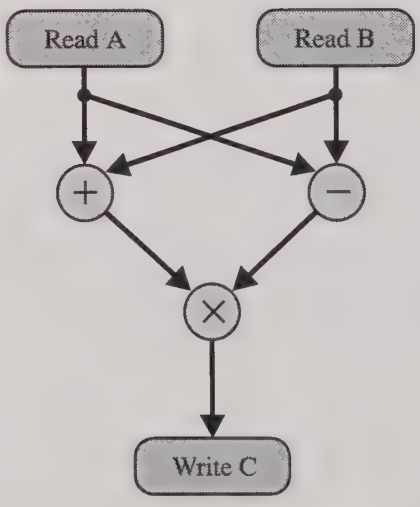

Figure 12.1: Data Dependency Graph

`READ A` and `READ B` operators. There is no longer need for intermediate variable names. Their existence is represented implicitly by the links between the + and − operations and the × operation. In fact, if A and B are results from some other graph, then even their names are unnecessary.

12.2 Data-Flow and WiT Igraphs

In WiT, you can install multiple servers (of potentially different architectures consisting of special hardware, more about this later), in which case the internal scheduler in WiT will take care to keep as many of the servers as busy as possible, thus fully utilizing your networked resources and the inherent parallelism in your algorithm.

When an Igraph starts running, only operators that do not require any input tokens, such as **rdObj**, are ready for execution. After an operator finishes processing its inputs, it may generate one or more output tokens, which will be sent along the links down the execution chain, and the input tokens will be consumed. A token can be sent to the input port of a following operator only if that port is empty. If an operator cannot send all its outputs successfully to all its descendants because some of the ports are full, it is blocked until the descendants are ready to receive the new tokens.

An Igraph execution is considered complete if there are no more operators ready for execution and all the servers are idle, which means that no tokens can be produced that may unblock any blocked operators. With animation turned on (**walk** or **run** speed), you can easily see where the Igraph execution is blocked (if it is) by the location of any unconsumed tokens.

12.3 The Scheduler

WiT uses a scheduler to decide which operators to run on which servers to maximize parallelism but maintain correct data-flow. Some operators are always run on the GUI program. These include all operators in the **dataflow** and **interactive** libraries. **Dataflow** library operators must run on the GUI because they are intimately tied to the scheduler. **Interactive** library operators either need access to the GUI display, or are fine grain operations which take little time to execute.

12.3.1 Data transport

Token data generally are not transported between the GUI and servers or among the servers themselves unless it is necessary. Often an entire algorithm, including input/output operations, can be executed on a single server without *any* data transfer at all. Data transfer is necessary when the object needs to be displayed, or when the server that has data does not support a required operation.

When multiple servers are involved, the scheduler maintains information about where each data object resides. When an operator needs to be scheduled, the scheduler assigns it to the available (not busy) server which requires the *least* amount of data transfer.

When an link branches into parallel branches, copies of the token data will be physically copied to multiple servers. But after this initial copying, the servers will be able to execute in parallel.

12.3.2 Firing Strategy

In general, operators are executed in the order in which they become ready. But if they become ready at the same time, then the scheduler is free to select whichever to execute first. Also, for efficiency and implementation reasons, UI operators are always scheduled before server operators, even if the server operators become ready before the UI operators.

An operator becomes ready whenever all its inputs are available (see Section 12.4.2 for exceptions). After execution, it tries to send all its outputs to its descendants. However, it may not be successful because some of its descendants may still be processing previous inputs, and are not ready to accept any new inputs. In this case the ancestor operator becomes blocked. When an operator has finished execution, it will clear its inputs and try to wake up any direct ancestors that were previously blocked.

You can have more than one output connected to the input of an operator. In this case, whichever token arrives first at the port will be accepted, and all other tokens will be blocked until the operator has consumed the first token.

12.3.3 Flat and Hierarchical Scheduling

The **Data-Driven Options** panel has a control for selecting between flat and hierarchical scheduling. This distinction only applies when an Igraph contains hierarchical operators.

In flat scheduling mode, the scheduler is free to choose any of the ready operators to execute, regardless of which Igraph the operators belong to. Hierarchical operators therefore act as though they are Igraphs shrunk to the size of icons. Not only do data tokens enter hierarchical operators and get executed immediately, outputs from hierarchical operators are sent downstream as soon as they become available.

In hierarchical scheduling mode, hierarchical operators behave as though they are primitive operators. In other words, data tokens tend to be collected before they get passed inside the underlying sub-graph to be processed. Also, output tokens are sent downstream only when all that can be produced from the sub-graph has been generated. This behavior is due to the fact that, in hierarchical scheduling, each sub-graph is considering as a distinct unit. Once scheduling has started in an Igraph, only operators from that Igraph are scheduled until there are no more operators to schedule from that Igraph.

The reason for providing two scheduling modes is that flat scheduling maximizes parallelism, whereas hierarchical scheduling executes functionally related operators in groups, which is important in generating readable program code.

12.4 WiT Extensions to Basic Data-Flow Model

In order to address the different requirements of diverse algorithms, WiT provides some useful variations to the basic data-flow model.

12.4.1 Unconnected Output Ports

If any output of an operator is not connected, then the operator will not block on these output ports. This makes it easier to construct Igraphs when you do not need one or more of the outputs from some of the operators. See the `sequence` Igraph for an example of unconnected outputs.

12.4.2 Fire-On-Any Operators

Some operators are best implemented if they fire as soon as *any* one of their inputs are ready, as opposed to waiting until all the inputs have arrived. An example is the **memory** operation in the **interactive** library. **Memory** takes either one of two inputs: an object and a flag. When the object input is fed to **memory**, it stores the object while sending the object to the output at the same time. If the flag input is fed, then **memory** recalls the stored object and sends it to the output.

If more than one input arrives at the same time to a fire-on-any operator, the tokens are consumed one by one, starting from the top port.

12.4.3 Parameter as Input on Fire-On-Any Operators

If a parameters on a fire-on-any operator is made into an input port, then for *each* actual input to the operator, a token must be supplied to the parameter port. This ensures that all parameters are properly set when a real input arrives.

12.4.4 Sync Tokens

When WiT is used to control external hardware, pure data-flow is not adequate. For example, often you need to initialize the hardware before it can be used. The initialization can be represented as WiT operators. Because hardware devices can often run in parallel, you would like to take advantage of the parallelism on the Igraph. So you would want the image processing to start only after *all* the initialization operators are finished.

You can create a null operator for this purpose. For example, if you have three hardware devices, you will have three parallel initialization branches, whose output you can send to a null operator that accepts three inputs to produce one output. The output is there simply to tell the rest of the Igraph that the hardware is now ready (we can depend on the fact that, in data-flow, an operator fires only after all its inputs have arrived). However, depending on the number of branches you want to merge, you will need null operators that accept different numbers of inputs. Our work will be easier and our Igraph more elegant if we can somehow tell a link junction (there are no limits to the number of links connected to a junction) to wait for all its input links to bring in a token before proceeding.

A special data token type called a **sync** token has been created for this purpose. **Sync** tokens do not carry any data, and they are treated specially by link junctions. If the tokens sent to a junction are **sync** tokens, then the junction will not propagate the token until *each* input link has brought in a **sync** token. If any of the links connected to the junction has been disabled (see Chapter 12), then

the junction will not wait for a **sync** token to come from the disabled link. In other words, disabled links behave as if they have been deleted as far as **sync** behavior is concerned.

When all the input links connected to a junction have brought in a **sync** token, the **sync** token (one only) is broadcast to all the output links connected to the junction.

You should never send **sync** tokens and regular data tokens to the same junction. There is no logical interpretation of how the junction should behave in such a construction.

12.5 Some Caveats of Data-Flow

Data-flow is a natural way of expressing parallel algorithms. However, most of us are much more familiar with the traditional control-flow model, where all data is stored in a commonly accessible area, and we specify the sequence of operations to be performed on the data. Almost all common programming languages, such as C, Pascal, or FORTRAN, work this way. In this paradigm, operations are performed on the data regardless of whether the data is valid or not. If the execution sequence is correct, then the data will be valid at the right time.

If you are not familiar with data-flow, you may find the behavior of WiT sometimes strange, but after you have a good understanding of the data-flow concept, you should find that it all makes sense and it is easy to construct data-flow algorithms in WiT. Some caveats of data-flow that you may encounter are explained in this section.

12.5.1 Deadlock

One of the first problems you may encounter while building your own Igraphs is that sometimes execution is blocked before the desired outputs are produced. For example, suppose we want to overlay a small image (say a company logo) onto a list of images. We can use the **sequencer** and the **dir** operator to form the image list, and use **aluOp** to do the overlay. Afterwards, we want to display the results, as illustrated in the Figure 12.2 (Igraph `deadlock` in demo directory).

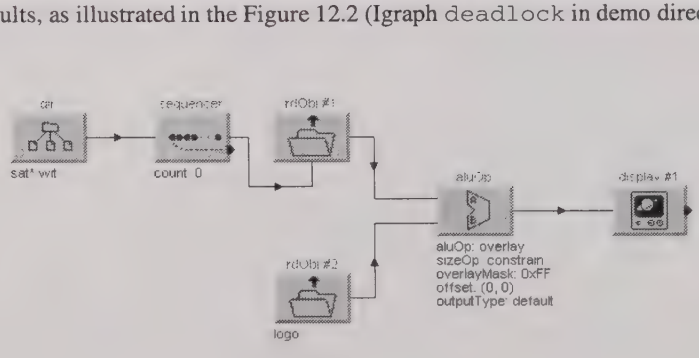

Figure 12.2: Example of Deadlock in an Igraph

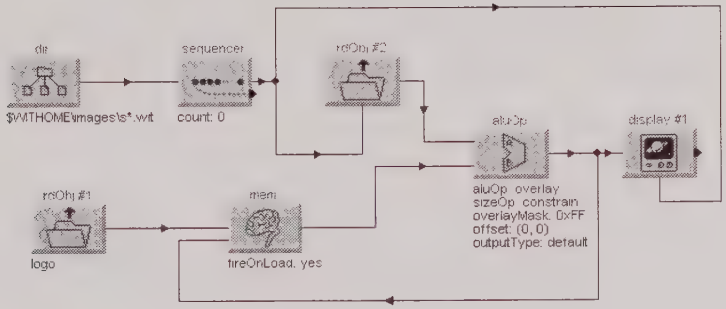

Figure 12.3: Deadlock Fixed

We used the file names as inputs to **display** so that the results will be shown in different windows.

However, this Igraph will stop after the first image is processed. With animation turned on, you will see that **aluOp** has one token waiting on the upper input. This means that **aluOp** is blocked because it needs a token on the lower port before it can fire. The logo token has been consumed when the first image was processed, and is no longer available. You may think that perhaps we should not consume input tokens, but this is the only way that an operator can signify to its ancestors that it is ready to process new tokens, and to eliminate unnecessary tokens, which require memory to store. The correct solution is to use the **memory** operator, which retains its output token so that it can be recalled later. The correct Igraph is shown in Figure 12.3 (Igraph `deadlockFixed` in demo directory).

12.5.2 Connecting an Operator to Itself

In general, loops are perfectly acceptable in an Igraph. However, you cannot connect an output of an operator directly (i.e. no other operators in the loop) to one of its own inputs. An example of such *unsupported* construct is shown in Figure 12.4. Such a construct is a problem because operator execution conceptually proceeds as follows:

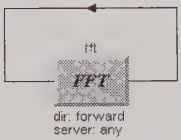

Figure 12.4: Illegal Igraph Constructs

1. The inputs are used to produce the outputs, if there are errors, exit.
2. Try to send the outputs to all descendants.
3. If successful, goto 4, else goto 2.
4. Delete the inputs and exit.

If you connect an output directly to one of the inputs, then step 2 will always fail, since a token can only be sent to a port if that port is empty, but we are sending the token to our own input ports, and the input ports are not cleared until step 4 is executed. In practice, you should not do this anyway since that would result in an infinite loop. If you really want an infinite loop (for use as a free running demo, for example), then you should do something in the loop. See the `if-demo` Igraph for an example.

Chapter 13

Configuration

The WiT work environment is user-configurable. Configurable options include the position and size of the main WiT window, where pop-up windows are placed, where servers, libraries, or icons are located on the file system, how many servers to run, customize operator icons, etc.

WiT configuration is classified into two categories:

- GUI initialization (§13.1).
- Server configuration (§13.3).

13.1 GUI Initialization

When WiT starts, GUI options (such as the position and size of the main workspace) are set with the following sequence (increasing priority):

1. Built-in default options (§13.1.2).
2. Options specified in the initialization file (§13.1.3). Override any default options.
3. Options specified on the command line (§13.1.4). Override any configuration file options.

13.1.1 Initialization Options List

Following is a list of all the initialization options and their meanings:

announce On stripped down versions of WiT, WiT makes an announcement which lasts for a few seconds on start-up. When you have become familiar with WiT, you will probably want to disable this message. Valid values: 0 (**disable**), 1 (**enable**).

autoclr Specifies whether pop-up object windows should be cleared automatically when an Igraph or demand history is run, or when a new Igraph or demand history file is loaded. Valid values: 0 (**off**), 1 (**on**).

13.1.4 Command Line Options

Command line options override any default or initialization file settings. The option names used on the command line is the same names as in the initialization file (§13.1.1), but with a dash ('-') in front.

For example, to set the execution mode to demand-driven, the number of demand-driven mode registers to 8, the WiT window to position (200,200) with a size of 400 × 300, and to load the history file 'myhist' on startup, use these command line options:

```
-mode demand -nreg 8 -window 200 200 400 300 -open myhist
```

13.2 Color

When WiT is run on 8 bits/pixel (256 color) displays, sharing of colors with other applications you may be running become an issue. These colors must be shared among all applications currently running, or else the system colormap will have to be swapped as you move your mouse into different applications, causing irritating sudden color changes. By default, WiT requests just enough colors from the window manager so that animation and Igraph editing (rubber banding, etc) can be nicely done and images can be shown with reasonable contrast, both grayscale and color. Specifically, WiT uses 8 colors for the Igraph and other graphics, 48 colors for grayscale images, and 64 colors for color images.

In the event that there are not enough colors available, WiT will switch to a private colormap. In order to minimize the color flashes associated with using private colormaps, WiT preserves the first 8 colors of the system colormap. These are the colors that the OpenLook window manager uses. This way even though colors will flash when you move your mouse in and out of WiT and other applications, at least the window frames and other vital graphics are always visible and shown without color flashes.

The default 48 colors used for grayscale images provide reasonably smooth rendition of grayscale images. However, if necessary you can increase the number of colors allocated with the **ngray** option (§13.1.1).

When WiT is run on color displays with more than 8 bits/pixel, all images can be displayed at the same time. The more colors your display can support, the more accurate the images will be displayed.

13.2.1 Workspace Colors

You can change the colors that WiT uses for the workspace, such as colors used for drawing icons, the workspace background, and the grid. To change a particular color, (e.g. the color used for operator parameters):

1. Select the **Options/Setup** menu item. The Setup panel (§10.4.1) comes up.

2. Select **Custom** for the **Workspace Colors** control. A color editing panel comes up (Figure 13.1). On the right of this panel is a set of radio buttons for object types (§13.2.1) whose colors can be changed. On the left is a color selection panel (§10.10).

3. Click the button for the object type whose color you want to change. E.g. **Parameter**.

4. Use the color disc or sliders to select a color

5. Repeat the previous two steps for other colors you want to change.

6. When you are satisfied with all the colors, click the **OK** button. Your customized colors is saved in the initialization file (§13.1.3).

If you want to abort all changes you have made and revert the original custom color definitions before you started, hit the **Reset** button instead of the **OK** button.

If you want to remove all records of your current custom color definitions, click the **Remove** button. The next time you edit custom colors, WiT will copy the current color scheme (**Light**, **Dark**, or **Color**) to the custom color scheme, and then you can make changes starting from those colors.

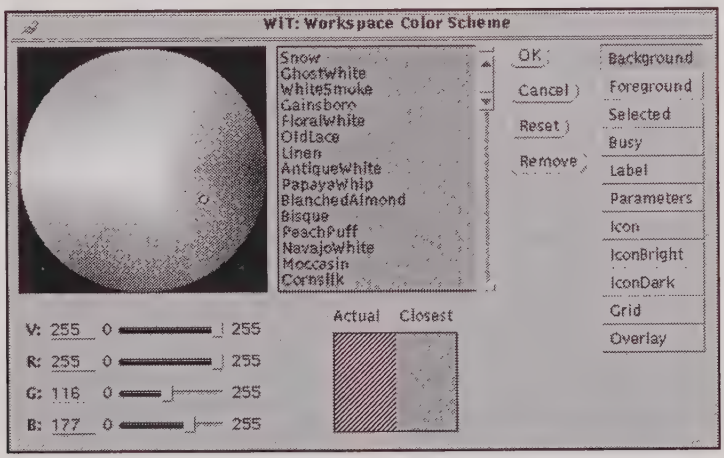

Figure 13.1: Color Editor for Workspace Colors

Workspace Object Color Types

Background	Background for Igraphs, operator panels, filer, etc.
Foreground	Links, probes, ports, and icon graphics.
Selected	Selected objects.
Busy	Operators currently being executed.
Labels	Operator names above icons.

Parameters	Parameter names and values below icons.
Icons	Background of icons.
IconBright	Top and left borders of icons (for 3D effect).
IconDark	Bottom and right borders of icons (for 3D effect).
Grid	Grid on Igraph.
Overlay	Color used when objects are dragged.

13.3 Server Configuration

Information about server configuration such as how many servers to run, search paths for WiT objects and Igraphs, what operators are supported on which server, is stored in a configuration file, which has a 'wrc' file extension. You can have different configuration files for different uses of WiT. For example, you can have a configuration file for only one local server for when you are designing algorithms and speed is not a major concern. But you may want to have another configuration, to be used at times when most of the computers in your work place are idle, which uses multiple servers (§13.6) to maximizes computing power.

13.3.1 Choosing a WiT Configuration

Configuration files are normally kept in the `$WITHOME/config` directory, although they can be stored anywhere. Choosing configurations is done using the **Server Manager** (§11).

13.3.2 Creating a New Configuration File

To create a new configuration file:

1. Choose an existing configuration file closest to what you want, and copy it to a new name. For example, if you are building you own server with a new operator library, you can start with `win31.wrc` or `nt.wrc`. So you can copy it to a file named `myconfig.wrc`, for example.
2. Use a text editor to modify the configuration file.
3. Save the configuration file. The configuration can be used immediately (§13.3.1).

13.4 WiT Configuration File Format

A WiT configuration file contains (possibly nested) attribute/value pairs surrounded by angle brackets:

<attribute value>

The value portion may itself contain attribute/value pairs.

The attributes and their meanings are:

Attribute	Value
objpath	Search path for WiT data objects (§13.4.4).
igrpath	Search path for Igraphs (§13.4.5).
libpath	Search path for operator definitions, icons, and help files (§13.4.6).
svrpath	Search path for server include files and executables.
iconcache	Location of operator icon cache (§4.2.4).
server	Server type definition (§13.4.1).
operators	Operator list supported by a particular server (§13.4.2).

13.4.1 Server Type Definition

Each server definition must consist of *all* the following attributes:

binary	The name of the server executable, e.g. witwin31.
group	The group name, can be any string except 'ui'. The name 'ui' is reserved to mean the WiT GUI. This group name must correspond to one of the operator list definitions (§13.4.2).
arch	The architecture that the server is built with. E.g. sun4, win31, win32
hostname	The name of the computer where the server will run on. If the server is going to run on the same computer as WiT, then use the name **localhost**.
maxOp	A reserved attribute. Set the value to 20.
maxObj	A reserved attribute. Set the value to 20.

13.4.2 Operator List Definition

Each operator list definition consists of a group name attribute followed by any number of library attributes:

group	The group name, can be any string except 'ui'. The name 'ui' is reserved to mean the WiT GUI. This name must correspond to the group name of at least one of the server type names (§13.4.1).
define	Short library name followed by long library name (§13.4.3), followed by an optional **invisible** keyword, for libraries which do not need a graphical panel in WiT.

Unless you want to add new operators or server types, you do not need to modify the operator list definitions. Refer to the *Programmer's Guide* if you want more information.

13.4.3 Library Names

Each operator must have a long name and a short name. The long name is used by WiT to label the library panel, menu, etc. The short name is used for directory and file names under the library path (§13.4.6):

- def
- icons
- icons
- help

The short name is also used for WiT generated header files (see *Programmer's Guide*).

For example, the library **Data Manipulation** has a short name **data**. Its library path is `$WITHOME/lib`. Therefore, you will find these directories under it:

- `def/data`. The operator definition file (§4.2.1) for all operators in the **data** library. There *must* be an entry in this file for each operator in the **data** library.
- `.icons/data`. Each operator in the **point** library *may* have an icon file under this directory. Icon files must be in XBM format (§4.2.3). The name of the icon file must be same as the operator name, with no file extension. If an operator does not have an icon in this directory, then the default icon is used (§4.2.5).
- `.help/data`. Each operator in the **point** library *may* have a help file under this directory. Help files must be text files. The name of the help file must be same as the operator name, with no file extension.

13.4.4 Object Search Path

The object search path specifies where objects can be found. The value is a colon (;) separated list of directories, like the syntax used to specify search paths in MS-DOS. You can use environment variables in the paths, such as `${WITHOME}` and `${HOME}`. You can also use dot (`.`) and double dot (`..`) to specify the current directory and the parent directory respectively. Since the configuration file is parsed when you start WiT, the current directory that dot refers to will be the directory in which you start WiT.

When WiT reads an object, the `objpath` is searched for the object name. Searching is performed from left to right. So for the example `objpath`, the current directory (.) will be searched first for the object. If found, it will be loaded. If not, the searching continues with the next directory on the path list until either the object is found or the directories are exhausted.

13.4.5 Igraph Search Path

The Igraph search path works similarly as the object search path (§13.4.4). It is used to locate Igraphs associated with hierarchical operators. Notice that since a top-level Igraph must be loaded from the filer, its path is already uniquely specified and no searching is necessary.

13.4.6 Library Search Path

The library search path works similarly as the object search path (§13.4.4). It is used to specify where operator definitions, icons, and help files can be found.

13.4.7 Default Configuration File

The default configuration file as shipped looks like this:

```
#
# WIT uses search paths to lookup the location of
# objects (objpath), Igraphs (igrpath), operator libraries
# (libpath), and WIT servers (svrpath).
#
# A pathname is a directory which may utilize environment
# variables. Each pathname is separated by a colon.
# Any path entry may be used more than once in which case
# the collective set of paths are used starting from the
# search from the first definition.
#
<objpath      .;${WITHOME}/images>
<igrpath      .;${WITHOME}/lib/igr;${WITHOME}/demo>
<libpath      ${WITHOME}/lib;${WITHOME}/proto>
<svrpath      ${WITHOME}/servers>
#
# The iconcache defines a cache file for reading
# operator icons. When this entry is present, icons for operators
# are read from this file to accelerate the start up process.
# If new icons are introduced, then this file must first be
# removed to force an update of the cache.
#
<iconcache    ${HOME}/witicons>
#
# The server keyword defines a WiT server which is to be started
# and utilized by the UI. The set of nested keyword-value
# pairs describe the name of the server (binary), the type
# of operators the server can run (group), the server's
# architecture (arch), which machine the server will run
# on (hostname), and two attributes to be used in the future
# (maxOp and maxObj).
#
<server
      <binary         witsun4>
      <group          general>
      <arch           sun4>
      <hostname       localhost>
      <maxOp          200>
      <maxObj         20>
>
#
# The operators keyword defines a list of operator libraries
# belonging to a particular group. This group is then associated
# with a server. The set of nested
# keyword-value pairs defines the group name (group) and
# the various operator definition files (define). A define
# has two attributes, the name of the actual C library where the
# operators can be found and a descriptive name to be used
# by WiT in the operators pull-down menu.
#
<operators
```

```
        <group          ui>
        <define         uiLib           Interactive>
        <define         flowLib         Dataflow>
        >
<operators
        <group          general>
        <define         ioLib           Read/Write>
        <define         dataLib         DataManipulation>
        <define         pointLib        Point>
        <define         morphoLib       Morphology>
        <define         filterLib       Filters>
        <define         xformLib        Transforms>
        <define         measLib         Measurement>
        <define         segmentLib      Segmentation>
        <define         pyramidLib      Pyramids>
        <define         protoLib        Prototypes>
        >
```

13.5 Password Information

There is also a password file for machine authentication. The password file is used by the WiT floating license manager to determine if the machine WiT is started on is a valid machine based on an encryption of the `hostid`.

13.6 Servers

As shipped, WiT is configured to run all built-in operators within the WiT GUI. However, WiT is based on a server/client model where the client is the GUI part and the server(s) is comprised of one or many computational servers. The client and servers are separate programs, possibly running on different computers! If your computer is connected to a local area network, you may want to run WiT servers on all machines that are not busy, so that your Igraphs can execute faster if there is any inherent parallelism in them.

Even if your computer is not networked, the separation into client and server halves allows you to add your own operators to WiT by rebuilding the server part only. The client (GUI) part never needs to be changed. If you want to launch the server as a separate program, use the Server Manager panel (or program) to select the the `svr.wrc` configuration file (§13.3). This will give you a single detached server running on the same machine as the GUI. If you want to switch back to a no server configuration, select `gui.wrc` on the Server Manager.

If you want to start multiple servers, you will need to modify the configuration file. A separate `server` entry is needed for each server you want to run. In most cases, the user need only duplicate an existing `server` entry then change the machine name for where the server will run. For example, suppose we have a network which consists of four machines: `earth` running Windows NT, `mars` running Windows NT, `venus` running Windows 3.1, and `pluto` running SunOS. We want to run the WiT GUI on `earth`, with a WiT server running on each of `earth`, `mars`, `venus`, and `pluto`. The server on `earth` is a local server using DDE to communicate with the GUI. The others use sockets to communicate. Our configuration file should look like this:

```
<objpath        ...
    ...
    ...
<server
```

```
        <binary         witnt>
        <group          general>
        <arch           win32>
        <ipc            dde>
        <hostname       mars>
        <maxOp          200>
        <maxObj         10>
        >
<server
        <binary         witnt>
        <group          general>
        <arch           win32>
        <ipc            socket>
        <hostname       mars>
        <maxOp          200>
        <maxObj         10>
        >
<server
        <binary         witnt>
        <group          general>
        <arch           win31>
        <ipc            socket>
        <hostname       venus>
        <maxOp          200>
        <maxObj         10>
        >
<server
        <binary         witsun4>
        <group          general>
        <arch           sun4>
        <ipc            socket>
        <hostname       pluto>
        <maxOp          200>
        <maxObj         10>
        >
<operators
    ...
    ...
```

13.6.1 Multiple server types

When the same operator name appears for different server types in the configuration file, a server type property in the parameter panel will appear when the operator is selected (Figure 13.2). This permits the user to decide where an operator is to be executed. The icon for the duplicate operator will appear in the first operator library panel. For example, consider the case where you have two servers called **witsparc** and **witi860** where the former is a general server and the latter is a server supporting the Intel i860 processor. If the **rdObj** operator is defined in the **Read/Write** library which is included in both the general and i860 servers then the property panel for the **rdObj** operator would appear as in Figure 13.2.

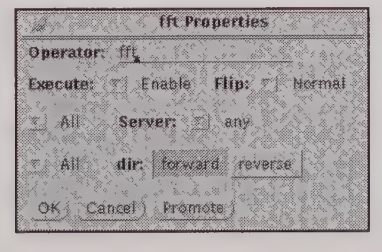

Figure 13.2: Multiple Server Types

There is a new item called **Server** on this property panel. Using our example, you can select either **witsparc** or **witi860**, and WiT will schedule the operator *only* on the specified server. If the specified server is busy but the other server is available, WiT will still wait for the specified server to become available. If you do not care which server the operator runs on, you can select **any**. When **any** is selected and all the servers are idle, the server whose name appears first in the **Server** list will be selected.

13.6.2 Multiple Users

WiT is an open-ended system. You can add your own operators, change icons, create new libraries, add new object types, and even develop new servers. Some of these you may want to share with other users, and others you may want to keep private. On the other hand, most users (at least when they first start) will tend to use only the built-in functionality that WiT offers. We have attempted to provide a system that is simple for novice users, and yet does not compromise on the flexibility that experienced users require.

WiT is easy for new users to use. You just set the `$WITHOME` environment variable, choose a directory for your project, start WiT from there, and do your image processing. WiT stores all the Igraphs and objects you create in this working directory. Navigating among several working directories should also present no problems, through the use of the user-friendly **File** panel.

The next step in the learning curve is probably to be able to access files automatically from different locations. For example, you might have created some useful Igraphs and objects for one project that you want to access from different directories associated with another project. This, we have seen, can be done by modifying the `objpath` attribute in the `witrc` file.

So far, all the customization does not affect other WiT users. However, as you become more familiar with WiT, you will likely want to develop your own operators. Or, if you are artistic, you may think of better icon designs that are more intuitive for you. So you would want to change the built-in icon graphics. Such changes, if not properly taken care of, *will* affect other users. Some users may want to use your new operators or icons, while others may not appreciate any unwanted modifications to their environment.

Adding operators, objects, or server types are advanced topics and are discussed in full detail in the *Programmer's Guide*. Here we will just mention how such changes can be easily and effectively controlled through the `witrc` file.

If you are going to modify any libraries (whether it be new operators, new icons, or new help files), you have to first make a copy of the default library (e.g. `$WITHOME/lib`), and change the `libpath` attribute in `witrc`. Any changes you make to your *private* library therefore will not affect anybody else. For users who want to use your library, they can set their `libpath` to point to your library. If you write-protect your library, then you can make sure these users will only use your library, but not modify it. If you want to add a new server type (for example, a server that supports some special purpose hardware), then you modify the `svrpath` attribute in `witrc`.

13.7 Directory Organization

The WiT distribution files are organized as follows:

`bin`		WiT executables.
`config`		Default icon, witrc, and iconcache. On-line help files.
`demo`		Example Igraphs.
`images`		Collection of 8-bit, 16-bit, float, and color images.
`h`		Header files for object/operator/server development.
`lib`		WiT operator development directory containing operator and object definition files, icons, help files, include files, and libraries.
	`def`	Operator and object definition files.
	`icons`	Operator icon files.
	`help`	Operator help files.
	`h`	Operator and object header files.
	`src`	Operator source directory for Prototype library.
	arch	Prebuilt libraries for your architecture.
`servers`		Server directory containing executables and interface source code.
	`bin`	Server executables.
	`src`	Server interface source code.
	`wit`*arch*	Server build directory.
`samples`		A WiT development directory example setup.

13.8 Read/Write Directories

When WiT reads something from the file system, such as an Igraph or an image, it goes through a list of directories (search paths) (§13.4) to locate it. The first file that has the correct name is used.

When WiT writes out something, only two possible directories are used: an Igraph directory for Igraphs and demand history files, and an object directory for objects. The Igraph directory is set when you select either the **File/Load** and **File/Save as** menu items. The object directory is set when you select the **File/Object dir** menu item.

One exception to this rule is when an Igraph has been read from the file system (as opposed to having been designed from scratch). If the Igraph is saved using the **File/Save** menu item, the Igraph will be saved to where it originally came from. If you do not want this, use **File/Save as** instead. This will allow you to choose both a new name for the Igraph and the directory to which it is saved. The directory then becomes the default Igraph directory.

Both Igraph and object directories default to the directory that you start up WiT initially. You can override these defaults with command line options (§13.1.4) or from the initialization file (§13.1.3).

Index

accuracy, *see* tolerance
adjust
    object, 72
animation, 119, 123
array editor, 102
attribute
    environment, 130

block
    Igraph, 122
    operator, 119
bypass, 101

choice list, 97
color, 128
color panel, 98
compiler, 83
cut, *see* delete

data-driven, 117
data-flow
    extensions, 120
deadlock, 122
deselect
    object, 71

file panel, 96
font, 73
font panel, 97

getData, 75
graph, 79
gravity, *see* tolerance
group properties, 87

hardcopy, 73, 111

icon
    flip, 101
    icon editor, 104
Igraph
    run, 87
image
    colormap, 107, 108
    index, 106
    profile, 106
    property, 105
    quality, 112
    save, 108
    vector, 106
INI file, 127
initialization file, 127

link
    properties, 102
    thickness, 102

main window, 85
menu bar, 86
modify
    object, 72
move
    object, 71

node, *see* operator

operator
    block, 119
    data-driven properties, 100
    demand-driven properties, 100
    disable, 101
    fire-on-any, 121
    flip, 101
    modifying icon, 104
    modifying port positions, 104
options
    command line, 128
    default, 126

initialization file, 127
names, 125
priority, 125

pan, 70
parameter
display, 102
port
unconnected, 121
PostScript, 73
print, 73, 111
property, *see* parameter

remove, *see* delete
resize windows, 70
restore
from zoom and pan, 70
run
demand script, 87
Igraph, 87

scheduler, 119
data transport, 119
firing strategy, 120
flat, 120
hierarchical, 120
select
add, 71
area, 71
object, 71
window, 73
server/client, 134
shift key, 69
speed, 92
surface, 77
sync, 121
token, 121
system
operator, 82

token, 119–121, 123, 124
tool bar, 89
toolbar
button, 89
status indicator, 89
trace, 88

workspace
zoom, 69

zoom, 69
area, 69
zoom/pan panel, 99

www.manualsplus.com
1011009